Une introduction à Maple

Philippe Dumas, Xavier Gourdon

INRIA

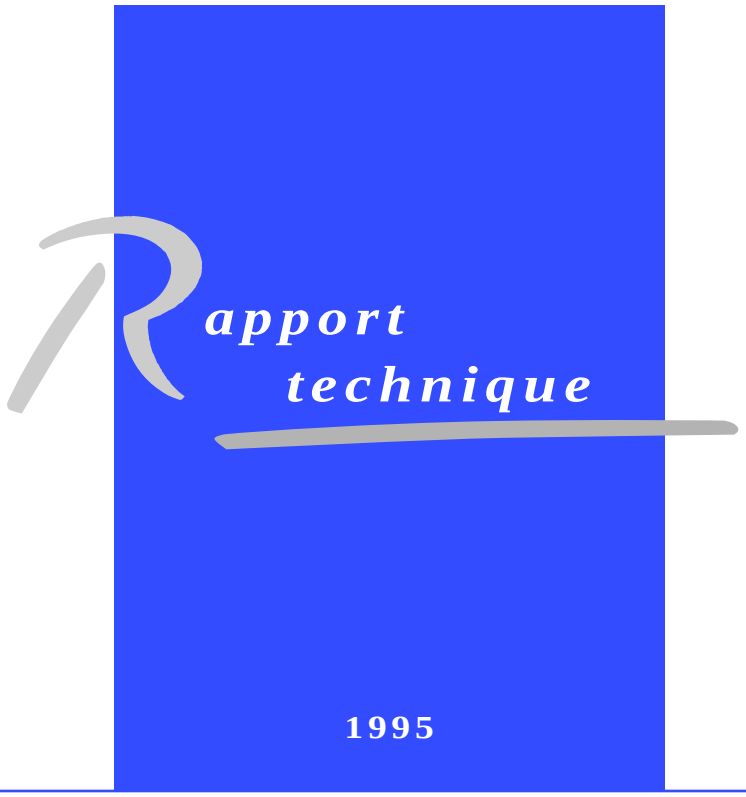
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

Une introduction à Maple

Philippe Dumas
Xavier Gourdon

N ° 173
Juin 1995

PROGRAMME 2



ISSN 0249-6399

Une introduction à MAPLE

Philippe Dumas et Xavier Gourdon

Résumé

Nous présentons le logiciel de calcul formel MAPLE sans en supposer aucune connaissance et décrivons ses fonctionnalités essentielles dans le but de rendre le lecteur autonome. La structure des objets et l'écriture des programmes sont illustrées d'exemples détaillés.

An introduction to MAPLE

Abstract

This is an introduction to the computer algebra system MAPLE. All the basic elements are presented so that the reader may become self-sufficient in using the system. Data structures and programming techniques are explained using detailed examples.

Des versions préliminaires de ce texte ont servi de support de cours à l'occasion de stages d'initiation MAPLE organisés par l'École des Mines de Nantes.

Une introduction à Maple

Ph. Dumas et X. Gourdon[1]

Résumé : *Nous présentons le logiciel de calcul formel* MAPLE *sans en supposer aucune connaissance et décrivons ses fonctionnalités essentielles dans le but de rendre le lecteur autonome. La structure des objets et l'écriture des programmes sont illustrées d'exemples détaillés.*

Le calcul scientifique sur ordinateur s'est longtemps identifié au calcul numérique. Cette vision est maintenant dépassée car depuis une vingtaine d'années il existe des systèmes de calcul formel qui permettent non seulement un traitement numérique mais aussi un traitement algébrique des quantités que l'on rencontre usuellement en mathématiques ou en physique. Ces systèmes, d'abord peu puissants et réservés aux initiés, ont maintenant une présentation conviviale et un emploi aisé qui les mettent à la portée de tout scientifique. Nous proposons ici une présentation succinte de l'un d'entre eux, le logiciel MAPLE développé à l'université de Waterloo (Canada). Le texte s'articule en quatre parties ; d'abord une excursion superficielle (section 1) ; puis une description du champ mathématique immédiatement abordable et qui permet déjà le traitement de nombreux problèmes (sections 2 et 3) ; ensuite nous pénétrons plus avant dans le système dans le souci de mieux comprendre son fonctionnement (sections 4 à 6) ; enfin nous proposons quelques thèmes d'étude qui permettent d'appliquer les notions acquises (section 7).

[1]Ce travail a été réalisé au sein du PROJET ALGORITHMES de l'INRIA. Nous remercions les membres du PROJET ALGORITHMES, notamment Bruno Salvy (INRIA) et Jacques Carette (Waterloo Maple Software), pour l'aide qu'ils nous ont apportée. Il est aussi le fruit de plusieurs stages d'initiation au logiciel MAPLE destinés à des professeurs de classes préparatoires pris en charge par l'École des Mines de Nantes.

Adresse postale: Projet ALGORITHMES, INRIA Rocquencourt, BP 105, 78153 Le Chesnay Cedex.
Adresse e-mail: Philippe.Dumas@inria.fr, Xavier.Gourdon@inria.fr.

Table des matières

1. Mise en jambe

Cette première partie, assez informelle, a pour but de faire sentir le champ d'application et la syntaxe de Maple. Toutes les notions effleurées ici seront développées dans la suite.

Nous supposons que vous êtes installés devant une station de travail et qu'une fenêtre Maple est ouverte. En haut de la fenêtre vous lisez le numéro de la version que vous utilisez ; il y a quelques chances que ce soit la version V.2 ou la version V.3. L'une ou l'autre convient pour ce que nous allons faire. Dans la fenêtre elle-même, vous voyez l'invite, le signe >, c'est-à-dire le caractère qui signale que Maple attend votre bon vouloir. Nous vous suggérons de taper les commandes qui suivent pour découvrir la syntaxe de Maple et mémoriser quelques commandes de base.

Commençons par le test usuel pour voir si tout est normal dans ce monde (il faut taper un retour chariot, `Return`, au bout de chaque ligne, ce qui est sous-entendu dans toute la suite).

```
> 1+1;
```

2

Remarquez le point virgule qui termine la commande ; on peut aussi utiliser le deux-points, mais alors le résultat n'est pas affiché. Si vous oubliez le point-virgule, tapez le à la ligne suivante ; les espacements ou les sauts de ligne supplémentaires n'ont pas d'importance.

```
> 1
>              +                  1
>                  ;
```

$$2$$

L'affectation se fait par le "deux points, égale".

```
> x:=Pi:x^2;evalf(",50);
```

$$\pi^2$$

$$9.8696044010893586188344909998761511353136994072408$$

Nous avons affecté la valeur π à x, puis affiché x^2 c'est-à-dire π^2 et enfin sa valeur numérique avec 50 chiffres décimaux. Au passage nous avons utilisé le caractère guillemet auquel est affecté le dernier résultat évalué. De même le pénultième résultat s'obtient par `""` et l'antépénultième par `"""`. Remarquez que Maple fait la différence entre majuscule et minuscule : la variable `pi` est différente de la variable prédéfinie `Pi`, même si sa représentation à l'écran est identique.

```
> """+1;
```

$$\pi + 1$$

Évidemment le fait d'avoir affecté une valeur à x empêche de l'utiliser maintenant comme variable mathématique ; pour le "désaffecter" il suffit de taper le nom de la variable entouré d'apostrophes (*quote*)

```
> x:='x';
```

$$x := x$$

```
> x^2;
```

$$x^2$$

Après ces quelques remarques entrons dans le vif du sujet. La syntaxe de Maple est proche de la syntaxe mathématique et on y dispose des fonctions usuelles comme les fonctions trigonométriques, l'exponentielle ou le logarithme. Voici le développement limité de la fonction tangente en 0 à l'ordre 14.

```
> series(tan(x),x,15);
```

$$x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + \frac{62}{2835}x^9 + \frac{1382}{155925}x^{11} + \frac{21844}{6081075}x^{13} + O\left(x^{15}\right)$$

Comme vous le voyez Maple utilise la notation O de Landau. Il s'agit d'une version affaiblie ; un $O(x^\alpha)$ représente en fait un $O(x^{\alpha+\epsilon})$ pour tout $\epsilon > 0$ comme on le voit dans l'exemple suivant où le $O(x^6)$ indiqué est en fait un $O(x^6 \ln(x)^6)$.

```
> series(x^x,x);
```

$$1 - \ln\left(\frac{1}{x}\right)x + \frac{1}{2}\ln\left(\frac{1}{x}\right)^2 x^2 - \frac{1}{6}\ln\left(\frac{1}{x}\right)^3 x^3 + \frac{1}{24}\ln\left(\frac{1}{x}\right)^4 x^4 - \frac{1}{120}\ln\left(\frac{1}{x}\right)^5 x^5 + O(x^6)$$

La commande

```
> series(tan(sin(x))-sin(tan(x)),x,8);
```

vous permet de résoudre un exercice éculé.

L'intégration se fait par `int`,

```
> int(1/(2+cos(t)),t);
```

$$\frac{2}{3}\sqrt{3}\arctan\left(\frac{1}{3}\tan\left(\frac{1}{2}t\right)\sqrt{3}\right)$$

on calcule une intégrale en spécifiant les bornes,

```
> int(exp(-u^4),u=0..infinity);
```

$$\frac{\pi\sqrt{2}}{4\,\Gamma(3/4)}$$

Remarquez que l'infini positif est **infinity** et que MAPLE connaît la fonction Gamma (et la formule des compléments). Si on veut une information plus précise sur la commande **int**, on ouvre une fenêtre d'aide par

```
> ?int
```

Une fenêtre apparaît alors, qui contient une description de la commande. Vous voyez en particulier des exemples à la fin de la description, ainsi que d'autres commandes MAPLE liées au sujet.

La différentiation se fait sans problème,

```
> diff(arctan(z)+arccot(z),z);
```

$$0$$

```
> diff((x-1)*(x-3)/(x^2+1),x);
```

$$\frac{x-3}{x^2+1}+\frac{x-1}{x^2+1}-\frac{(2\,x-2)\,(x-3)\,x}{(x^2+1)^2}$$

```
> normal(");
```

$$\frac{-4\,x+4\,x^2-4}{(x^2+1)^2}$$

```
> solve(",x);
```

$$\frac{\sqrt{5}}{2}+1/2,\,1/2-\frac{\sqrt{5}}{2}$$

Ici on a vérifié que $\mathrm{Arctan}\,z+\mathrm{Arccot}\,z$ est une constante, puis on a dérivé une fonction rationnelle, on a réduit au même dénominateur la dérivée et on a trouvé ses racines.

MAPLE sait résoudre certaines équations différentielles,

```
> equ:=4*diff(z(t),t$2)+diff(z(t),t)+4*z(t)=0;
```

$$equ := 4\,\frac{\partial^2}{\partial t^2}z(t)+\frac{\partial}{\partial t}z(t)+4\,z(t)=0$$

```
> dsolve(equ,z(t));
```

$$z(t)=\_C1\,e^{-\frac{t}{8}}\cos(\frac{3\sqrt{7}t}{8})+\_C2\,e^{-\frac{t}{8}}\sin(\frac{3\sqrt{7}t}{8})$$

La dérivée seconde aurait pu être notée **diff(z(t),t,t)**; on a employé le caractère **$** qui permet des répétitions. D'autre part les constantes introduites par le système ont un nom qui débute par le caractère de soulignement (*underscore*).

```
> dsolve(x*diff(y(x),x$2)+diff(y(x),x)+x*y(x),y(x));
```

$$y(x)=\_C1\,BesselJ(0,x)+\_C2\,BesselY(0,x)$$

Supposons que vous ayez une envie subite d'utiliser les polynômes de Legendre. Vous vous doutez qu'ils sont connus de MAPLE mais vous ne savez pas comment les obtenir. Dans le menu *Help* vous choisissez la commande *Keyword Search* (recherche par mot clé), puis dans la fenêtre qui apparaît, vous tapez **legendre** dans la case *Search For String* (recherche d'une chaîne de caractères). Vous obtenez une flopée de réponses, dont la dernière, **orthopoly,P**, est celle qui vous intéresse. Ceci dit, vous ne voyez pas clairement comment utiliser cette information. Essayons par une autre voie. Activez cette fois-ci la commande *Help Browser* [2] du menu *Help* . Dans la fenêtre ainsi ouverte, vous choisissez comme sujet *Mathematics* et vous parcourez

[2] *to browse through a book :* feuilleter un livre

les “sous-sujets” que l'on vous offre. *Special Functions* pourrait peut-être faire l'affaire ; vous cliquez sur ce *Subtopic*. On vous propose la fonction d'Airy, les fonctions de Bessel, ..., mais rien ne semble convenir. Peut être le dernier, *Integrals*, conviendrait ; en cliquant vous voyez apparaître *Legendre* mais la description dans la case inférieure vous montre qu'il s'agit d'intégrales elliptiques. Revenons au premier *Subtopic* (la deuxième petite fenêtre) et choisissons *Libraries and Packages*, puis dans le choix proposé `orthopoly`. On nous dit qu'il y a un *package* `orthopoly`. En cliquant sur le *Help* qui est en bas de la fenêtre, on obtient une fenêtre d'aide (on pourrait aussi taper `?orthopoly` dans la fenêtre principale). La fenêtre d'aide décrit le *package* `orthopoly` et en particulier les polynômes de Legendre qui y sont notés `P`. On y explique comment utiliser le *package*. Si nous désirons seulement connaître le septième polynôme de Legendre, nous tapons

```
> orthopoly[P](7,z);
```

$$\frac{429\,z^7}{16}-\frac{693\,z^5}{16}+\frac{315\,z^3}{16}-\frac{35\,z}{16}$$

Pour une utilisation plus intensive du *package* nous remplaçons les noms longs `orthopoly[joli_nom]` par les noms courts `joli_nom` en utilisant l'instruction

```
> with(orthopoly);
```

$$[G, H, L, P, T, U]$$

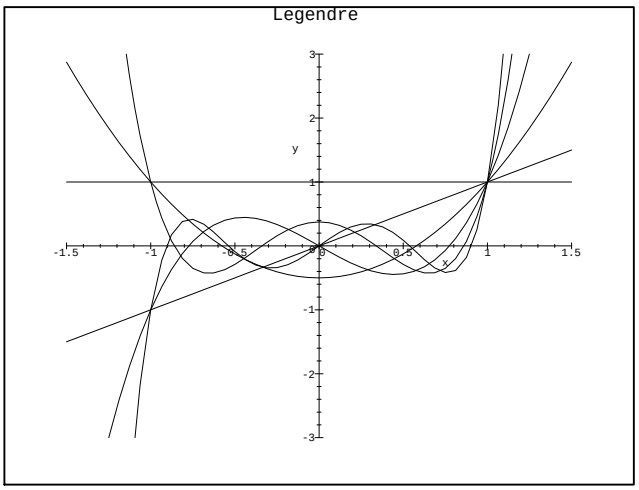
```
> P(7,z);
```

$$\frac{429\,z^7}{16}-\frac{693\,z^5}{16}+\frac{315\,z^3}{16}-\frac{35\,z}{16}$$

Maintenant que nous disposons des polynômes de Legendre, regardons les.

```
> plot({seq(P(n,x),n=0..5)},x=-1.5..1.5,y=-3..3,title=‘Legendre‘);
```

Après quelques instants une fenêtre s'ouvre et vous voyez les graphiques des P_n pour n allant de 0 à 5.



Expliquons un peu la commande employée ; `seq` permet de construire une séquence

```
> seq(i*(i+1)/2,i=0..10);
```

$$0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55$$

en l'entourant d'accolades (*braces*) on en fait un ensemble, alors qu'en l'entourant de crochets (*brackets*) on en fait une liste. Ci-après nous créons une liste et nous écrivons la liste renversée.

```
> T:=[];
```

$$T := [0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55]$$

```
> N:=nops(T);[seq(T[N-i+1],i=1..N)];
```

$$N := 11$$

$$[55, 45, 36, 28, 21, 15, 10, 6, 3, 1, 0]$$

D'autre part le titre obtenu grâce à la commande optionnelle **title** est donné par la chaîne de caractères **‘Legendre‘**. On reconnaît qu'il s'agit d'une chaîne de caractères par la présence des caractères **‘** (*backquote*). Il ne faut pas confondre *backquote*, qui délimite les chaînes de caractères, et *quote*, que nous avons déjà rencontré et qui inhibe l'évaluation.

Revenons au dessin. Nous constatons avec délice que tous les zéros des polynômes sont dans $]-1, 1[$ et s'entrelacent aimablement. Pour nous en convaincre encore plus nous les calculons numériquement avec la commande **fsolve**.

```
> for n from 0 to 5 do sort([fsolve(P(n,x),x)]) od;
```

$$[]$$

$$[0]$$

$$[-0.5773502692, 0.5773502692]$$

$$[-0.7745966692, 0, 0.7745966692]$$

$$[-0.8611363116, -0.3399810436, 0.3399810436, 0.8611363116]$$

$$[-0.9061798459, -0.5384693101, 0, 0.5384693101, 0.9061798459]$$

Pour bien constater l'entrelacement nous avons trié les racines par la commande **sort**, dont vous pouvez avoir la description en tapant **?sort**.

Les polynômes P_0, P_1, ..., P_5 forment une base de l'espace des polynômes de degré inférieur ou égal à 5. Nous disposons de leur expression dans la base canonique 1, x, ..., x^5. Inversement on peut vouloir exprimer la base canonique dans la base des polynômes de Legendre. Pour ce faire nous écrivons la matrice dont les vecteurs colonnes sont constitués des coefficients des P_n, à l'aide de la commande **matrix** du *package* d'algèbre linéaire **linalg**.

```
> M:=linalg[matrix]([seq([seq(coeff(P(j,x),x,i),j=0..5)],i=0..5)]);
```

$$\begin{bmatrix} 1 & 0 & -1/2 & 0 & 3/8 & 0 \\ 0 & 1 & 0 & -3/2 & 0 & \frac{15}{8} \\ 0 & 0 & 3/2 & 0 & -\frac{15}{4} & 0 \\ 0 & 0 & 0 & 5/2 & 0 & -\frac{35}{4} \\ 0 & 0 & 0 & 0 & \frac{35}{8} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{63}{8} \end{bmatrix}$$

Ensuite nous calculons l'inverse de cette matrice.

```
> linalg[inverse](M);
```

$$\begin{bmatrix} 1 & 0 & 1/3 & 0 & 1/5 & 0 \\ 0 & 1 & 0 & 3/5 & 0 & 3/7 \\ 0 & 0 & 2/3 & 0 & 4/7 & 0 \\ 0 & 0 & 0 & 2/5 & 0 & 4/9 \\ 0 & 0 & 0 & 0 & \frac{8}{35} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{8}{63} \end{bmatrix}$$

Après avoir traité un problème à l'aide de MAPLE, il est naturel de vouloir sauver les résultats obtenus et le chemin suivi. L'option *Save as ...* du menu *File* permet de créer un fichier `toto.ms` qui contient disons l'apparence de la session MAPLE que l'on vient d'effectuer. On n'a cependant pas sauvé l'état du système en procédant ainsi. Il faut pour cela choisir l'option *Save kernel state* pour créer un fichier `toto.m` qui contient l'état de la session. Si on n'a pas sauvé l'état du système il suffira de reparcourir la session depuis son début en validant chaque instruction par un retour chariot (on pourrait d'ailleurs ne valider que certaines instructions). Pour imprimer l'apparence de la session, on passe par l'option *Print* du menu *File*. Précisons que pour charger un fichier `.ms` on utilise la commande *Open* du menu *File*.

2. Programmation en Maple

Nous passons en revue les structures de base qui permettent d'écrire un programme. On notera que la syntaxe de MAPLE présente de nombreuses similitudes avec celle de Pascal. Cette impression est cependant assez superficielle.

2.1. Procédures. Un programme MAPLE est essentiellement composé de *procédures*. Le squelette d'une procédure est

```
proc(paramètres de la procédure)
  local liste des variables locales;
  global liste des variables globales;
  options liste des options;
    commande 1;
    ...
    commande n
end;
```

Les lignes contenant les fonctions `local`, `global` et `options` peuvent être omises ; nous discuterons de leur utilisation plus loin. Une procédure renvoie toujours un résultat qui est celui de la dernière évaluation effectuée par MAPLE dans la procédure.

À titre d'exemple, voici une procédure que nous avons appelée `logdiff` qui renvoie la dérivée logarithmique d'une expression par rapport à x. Sous la session MAPLE, tapez

```
> logdiff:=proc(expr,x)
> diff(expr,x)/expr
> end:
```

Remarquez qu'il n'est pas nécessaire de mettre un point-virgule à la fin de la première ligne et de l'avant dernière, parce que `proc` et `end` font office de parenthèses respectivement ouvrante et fermante. On a le même phénomène avec `do` et `od` ou `if` et `fi`, que nous verrons bientôt. Tapez maintenant

```
> logdiff(a*x^alpha,x);
```

et vous constatez que MAPLE renvoie

$$\frac{\alpha}{x}.$$

Lorsque la procédure définit en fait une fonction par une expression comme dans le cas précédent, on peut préférer la forme fléchée

```
> logdiff2:= (expr,x)-> diff(expr,x)/expr:
> logdiff2(exp(alpha*x)*x^beta,x);
```

$$\frac{\alpha\,\mathrm{e}^{(\alpha x)}\,x^{\beta}+\frac{\mathrm{e}^{(\alpha x)}\,x^{\beta}}{x}}{\mathrm{e}^{(\alpha x)}\,x^{\beta}}.$$

2.2. Les primitives de programmation.

Les instructions répétitives ou conditionnelles. La syntaxe de ces instructions est proche de la syntaxe Pascal. Voici par exemple une exécution de l'algorithme d'Euclide controlée par une boucle `while`.

```
> a:=x^5-1; b:=x^2-1; while(b<>0) do r:=rem(a,b,x); a:=b; b:=r; od;
```

$$\begin{aligned} a &:= x^5-1\\ b &:= x^2-1\\ r &:= -1+x\\ a &:= x^2-1\\ b &:= -1+x\\ r &:= 0\\ a &:= -1+x\\ b &:= 0 \end{aligned}$$

Dans la condition qui suit le `while`, on peut utiliser les opérateurs logiques `and`, `or`, `not`, ainsi que les noms logiques `true` et `false`. Ces instructions pourraient être encapsulées dans une procédure de calcul de pgcd, qui existe d'ailleurs déjà sous le nom `gcd`.

```
euclide := proc(x,y,z)
  local a,b,r;          liste des variables locales à la procédure
    a:=x; b:=y;
    while b<>0 do       tant que b ≠ 0 ...
      r:=rem(a,b,z);    renvoie le reste de la division euclidienne de a par b
      a:=b;
      b:=r
    od;                 fin de la boucle while
  a                     résultat renvoyé par la procédure
end:
```

L'instruction `for` effectue une boucle contrôlée par une variable selon la syntaxe suivante :

```
for i from début by pas to fin while cond
do
  séquence d'instructions
od;
```

La variable i est initialisée à la valeur *début*, et la séquence d'instruction est répétée tant que $i \le$ *fin* (ou $i \ge$ *fin* si *pas* est strictement négatif) et que la condition *cond* a la valeur vraie, i étant changé en i+*pas* à la fin de chaque itération. Chacune des options `for`, `from`, `by`, `to` et `while` peuvent être omises ; par défaut, les paramètres *début* et *pas* sont égaux à 1.

Par exemple, le groupe d'instructions

```
> f:=1; to 3 do f:=1+int(subs(t=u,f)^2,u=0..t) od;
```

$$\begin{aligned} f &:= 1\\ f &:= 1+t\\ f &:= \frac{2}{3}+\frac{1}{3}(1+t)^3 \end{aligned}$$

$$f := 1 + t + \frac{2}{3}t^4 + t^2 + t^3 + \frac{1}{9}t^6 + \frac{1}{3}t^5 + \frac{1}{63}t^7$$

écrit les quatre premiers termes de la suite de polynômes (f_n) en la variable t, définie par $f_0 = 1$ et

$$f_{n+1}(t) = 1 + \int_0^t f_n(u)^2\, du$$

et renvoie celui d'indice 3,

```
> ";
```

$$1 + t + \frac{2}{3}t^4 + t^2 + t^3 + \frac{1}{9}t^6 + \frac{1}{3}t^5 + \frac{1}{63}t^7$$

Il est possible de combiner une boucle **for** avec l'instruction **while**. Par exemple, pour trouver le premier nombre premier strictement supérieur à 10^{10} on peut écrire

```
> for i from 10^10+1 by 2 while not isprime(i) do od;
> i;
```

$$10000000019$$

et en sortie, la valeur de i contient le nombre recherché, que l'on obtient d'ailleurs directement par la commande **nextprime(10^10)**,

```
> nextprime(10^10);nextprime(");
```

$$10000000019$$

$$10000000033$$

L'exécution conditionnelle s'écrit

if *condition* **then**
  séquence d'instructions 1
else
  séquence d'instructions 2
fi;

Elle a pour effet d'exécuter la *séquence d'instructions 1* si la condition est vraie, la *séquence d'instructions 2* sinon. On peut traiter plusieurs conditions en utilisant la commande **elif**. La procédure suivante calcule les termes de la suite du pliage de papier. On fabrique cette suite en pliant une infinité de fois et toujours dans le même sens une bande de papier, puis en la dépliant et en considérant les plis qui sont entrants ou sortants ; le codage des plis par 0 ou 1 suivant leur nature fournit la suite.

```
> papier:=proc(n) option remember;
    if (n mod 2)=1 then papier((n-1)/2)
    elif (n mod 4)=0 then 1
    else 0
    fi
  end:
> seq(papier(n),n=0..20);
```

$$1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1$$

L'option `remember`. Lorsqu'une procédure possède l'option `remember`, MAPLE stocke dans une table la valeur renvoyée par la procédure après chaque appel de cette dernière. Nous avons déjà utilisé cette optimisation dans la procédure précédente. Ainsi à chaque appel de la procédure, MAPLE commence par rechercher dans sa table de `remember` si la valeur de la procédure pour les paramètres entrés n'est pas déjà connue, ce qui permet d'éviter des calculs redondants. Pour illustrer cette option, voici deux procédures récursives qui calculent les nombres de Fibonacci :

```
fib1:=proc(n)                    fib2:=proc(n)
  option remember;
  if n<2 then 1                    if n<2 then 1
  else fib1(n-1)+fib1(n-2)         else fib2(n-1)+fib2(n-2)
  fi                               fi
end:                             end:
```

Le temps de calcul de F_{10} est nettement plus petit pour la seconde procédure, qui possède l'option `remember`.

```
> t:=time(): fib1(20): time()-t;
```

3.650

```
> t:=time(): fib2(20): time()-t;
```

0.017

Les commandes `RETURN` et `ERROR`. Lorsque MAPLE rencontre l'instruction `RETURN`(*expr*) au sein d'une procédure, il sort de la procédure et renvoie l'expression spécifiée *expr*. La commande `ERROR(`*'message d'erreur'*`)` produit le même effet, à ceci près qu'elle stoppe le déroulement du programme et renvoie le message d'erreur spécifié. Il faut noter que `RETURN` et `ERROR` ont l'inconvénient de ralentir très légèrement la procédure et leur emploi doit correspondre à une nécessité. Il est souvent plus judicieux d'utiliser un `else`.

2.3. Entrées, sorties. Il est plus agréable d'écrire ses programmes sous un éditeur, par exemple EMACS, puis de charger le fichier correspondant dans la session MAPLE. Supposons que vous ayez tapé votre programme sous EMACS et que vous l'ayez sauvé dans un fichier `toto`. Revenez dans votre session MAPLE et tapez

```
> read toto;
```

ce qui aura pour effet de charger les procédures qui sont contenues dans votre fichier.

Ceci est d'ailleurs l'occasion de préciser le rôle des *quotes* et des *backquotes*. Dans l'exemple précédent `toto` est un nom ; on aurait tout aussi bien pu l'écrire avec des *backquotes* `` `toto` ``, mais ceci est inutile car `toto` et `` `toto` `` sont exactement le même nom. Les *backquotes* ne deviennent réellement utiles que si le nom comporte des caractères particuliers comme le point ou le *backslash* ; dans le premier cas l'absence de *backquotes* fait que `toto.mpl` est compris comme `totompl` car le point est le signe de la concaténation ; dans le second le *backslash* `\` est vu comme le signe de la division et `Maple/toto.mpl` est compris comme le quotient de `Maple` et `totompl`.

D'autre part les *quotes* ne sont à utiliser que pour prévenir une éventuelle affectation comme `toto:=2` ; en effet dans la commande `read 'toto'` la présence des *quotes* bloquent l'évaluation et le `read` porte bien sur le nom `toto` et non sur la valeur associée. Ainsi un esprit inquiet ou un programmeur MAPLE dont les programmes vont avoir des utilisateurs non prévenus devrait écrire `read ''`` `toto` `''` pour faire lire le fichier de nom `toto`.

Nous avons déjà évoqué plus haut la sauvegarde des sessions de travail. On peut aussi sauver les affectations par la commande `save`. Signalons qu'il est possible de rediriger la sortie vers un fichier en utilisant `writeto` ; ceci peut être utile pour un résultat de grande taille que l'on veut conserver comme une session de *debuggage*.

`iquo(m,n)`	quotient de la division euclidienne de m par n
`irem(m,n)`	reste de la division euclidienne de m par n
`ifactor(n)`	factorisation de l'entier n
`numer`, `denom`	numérateur et dénominateur d'une fraction
`trunc`, `round`, `frac`, `floor`, `ceil`	parties entières ou fractionnaire
`evalf`	évaluation numérique en flottant
`evalc`	évaluation d'un complexe

TABLEAU 1. Les procédures usuelles sur les nombres.

3. OBJETS MATHÉMATIQUES EN MAPLE

Tous les calculs mathématiques de base peuvent être traités par MAPLE et nous en donnons un aperçu, en insistant plus particulièrement sur l'algèbre linéaire.

3.1. Entiers, rationnels, flottants, complexes. MAPLE peut gérer des entiers dont la taille atteint 500000 chiffres décimaux[3]. Le nombre de zéros par lequel se termine l'écriture décimale de 100! devient une évidence. On notera le troisième argument optionnel de `irem` qui permet d'utiliser le quotient de la division.

```
> N:=100!; for i  while irem(N,10,'q')=0 do N:=q od: i-1;
```

$$N := 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000$$

$$24$$

Les procédures les plus courantes sur les entiers sont préfixées par la lettre `i` comme on le voit dans le tableau 1. MAPLE simplifie systématiquement les nombres rationnels.

```
> x:=1/21+3/35;
```

$$x := \frac{2}{15}$$

La précision des nombres flottants (en nombre de chiffres) est contrôlée par la variable d'environnement `Digits`. Par défaut, `Digits` vaut 10.

```
> 1.1^20;
```

$$6.727499949$$

```
> Digits:=20: 1.1^20;
```

$$6.7274999493256000920$$

Une expression peut être évaluée en flottant grâce à la commande `evalf`.

```
> Digits:=30: x:=exp(Pi*sqrt(163)); evalf(x);
```

$$x := e^{(\pi\sqrt{163})}$$

$$262537412640768743.999999999999$$

```
> evalf(x,40);
```

$$262537412640768743.9999999999992500725972$$

Les nombres flottants sont "contagieux" en ce sens qu'une expression comportant un nombre flottant devient facilement flottante :

```
> 1.+1/35;
```

$$1.028571429$$

Le nombre complexe $i = \sqrt{-1}$ est représenté par `I` en MAPLE.

```
> z:=(-1-2*I)^2;
```

[3]Plus précisément, $2^19 = 524288$ chiffres sur les architectures 32 bits.

`subs(x=u,f)`	substitution de x par u dans l'expression f
`diff(f,x)`	dérivée de f par rapport à x
`int(f,x)`	primitive de f par rapport à x
`int(f,x=a..b)`	intégrale de f entre a et b
`series(f,x=a,n)`	développement de Taylor de f en x au voisinage de a à l'ordre n

TABLEAU 2. Les procédures courantes sur les expressions de fonctions.

$$z := -3 + 4I$$

Pour mettre un nombre complexe sous forme cartésienne, on utilise `evalc` :

```
> z^(1/2);
```

$$\sqrt{-3+4I}$$

```
> evalc(");
```

$$1 + 2I$$

MAPLE choisit ce qui apparaît comme la branche naturelle pour les fonctions multiformes mais l'utilisateur doit rester vigilant s'il veut composer de telles fonctions.

3.2. Fonctions définies par des expressions. En MAPLE les fonctions sont représentées par des expressions. Insistons sur ce point ; on ne passe pas à une procédure une fonction mais l'expression de cette fonction. En effet si l'on définit une fonction par une procédure

```
> f:= (x,y) -> x*sin(y)+y*sin(x):
```

la commande `diff(f,x)` renvoie 0 puisque `diff` s'applique au nom `f` de la procédure. Par contre, $f(x, y)$ représente l'expression à dériver.

```
> diff(f(x,y),x);
```

$$\sin(y) + y\cos(x)$$

Mais on voit qu'il était inutile de définir une procédure. On applique donc la règle suivante.

Les fonctions mathématiques sont représentées par des expressions.

La commande

```
> f:=x^2*sin(y)+cos(x^4)*y^2: g:=diff(f,x);
```

$$2x\sin(y) - 4\sin(x^4)x^3y^2$$

fournit la dérivée partielle $\partial f/\partial x$. Pour avoir la valeur de g en $(x, y) = (1, 1)$, on tape

```
> subs(x=1,y=1,g);
```

$$-2\sin(1)$$

Par contre on ne passe pas par les affectations

```
> x:=1:y:=1:
```

suivies de l'appel `> g;`. En effet, ce faisant, toutes les quantités liées à x et y se retrouvent modifiées dès qu'on les évalue.

```
> f;
```

$$\sin(1) + \cos(1)$$

```
> diff(g,x);
Error, wrong number (or type) of parameters in function diff
```

On pourrait évidemment désaffecter x et y, comme l'on dit, par la commande

`degree(p,x)`	degré du polynôme p en la variable x
`coeff(p,x,i)`	coefficient de x^i dans le polynôme p
`collect(p,x)`	regroupement des termes de p suivant les puissances de x
`expand(p)`	développement du polynôme p
`rem`	reste dans la division euclidienne
`quo`	quotient dans la division euclidienne
`factor`	factorisation dans le corps des coefficients
`gcd`	pcgd de deux polynômes
`solve`	procédure de résolution algébrique d'équations
`fsolve`	procédure de résolution numérique d'équations
`evala`	calcul sur des quantités algébriques
`normal`	réduction d'une fraction
`denom`, `numer`	dénominateur et numérateur d'une fraction
`convert(f,parfrac,x)`	décomposition de la fraction f en éléments simples

TABLEAU 3. Les procédures usuelles sur les polynômes et les fractions rationnelles.

```
> x:='x':y:='y':
```

ce qui leur redonnerait leur statut de variables mathématiques, mais outre que ceci manque singulièrement d'élégance, cette affectation suivie d'une désaffectation n'est pas une saine politique. Il vaut beaucoup mieux s'en tenir à la règle suivante.

> ***Une variable informatique est un nom dont la première occurrence apparaît dans le membre gauche d'une affectation.***
> ***Une variable mathématique est un nom dont la première occurrence apparaît dans le membre droit d'une affectation.***
> ***On n'affecte* jamais *une variable mathématique.***

Dans l'exemple précédent les noms `f` et `g` sont des variables informatiques, alors que les noms `x` et `y` sont des variables mathématiques.

3.3. Polynômes, fractions rationnelles. Dans tous les systèmes de calcul formel les polynômes forment le morceau de choix parce qu'ils sont l'archétype du calcul algébrique. On pourrait penser qu'ils ne forment qu'une petite partie des expressions algébriques courantes ; il n'en est rien car beaucoup d'expressions algébriques sont des polynômes en d'autres expressions. Illustrons l'emploi de quelques procédures.

On peut obtenir le coefficient multinomial

$$\binom{30}{10,10,10} = \frac{30!}{(10!)^3}$$

comme le coefficient de $x^{10}y^{10}$ dans $(1+x+y)^{30}$. L'emploi de `coeff` ou `degree` suppose que le polynôme est sous forme développée.

```
> p:=expand((1+x+y)^30): coeff( coeff(p,x,10) ,y,10);
                                  5550996791340
```

Notons que ce coefficient multinomial peut se calculer plus efficacement avec le *package* `combinat` (cf. `multinomial`).

On convertit un développement de Taylor en un polynôme grâce à la commande `convert`. Pour admirer le contact entre la fonction sinus et son développement en série à l'ordre 10, on peut taper

```
> p:=convert(series(sin(x),x,10),polynom):
> plot({sin(x),p},x=0..5);
```

Le dixième polynôme de Bernoulli B_{10} peut s'obtenir à partir de sa série génératrice par

```
> f:=z*exp(z*x)/(exp(z)-1): 10!*coeff(series(f,z,12),z,10);
```

$$-7\,x^6+\frac{5}{66}-\frac{3\,x^2}{2}+5\,x^4+x^{10}+\frac{15\,x^8}{2}-5\,x^9$$

On aurait pu utiliser la commande `bernoulli(10,z);`.

Les racines d'un polynôme sont obtenues avec la commande **solve**

```
> solve(x^2-x-1,x);
```

$$\frac{1}{2}+\frac{1}{2}\sqrt{5},\ \frac{1}{2}-\frac{1}{2}\sqrt{5}$$

En général, les équations polynomiales ne sont pas solubles par radicaux :

```
> p:=x^5+2*x+1: solve(p,x);
```

$$\mathrm{RootOf}(\_Z^5+2\,\_Z+1)$$

L'expression renvoyée est une représentation des racines du polynôme x^5+2x+1. Pour obtenir une approximation numérique des racines réelles (resp. complexes) de p, on utilise la commande `fsolve(p,x)` (resp. `fsolve(p,x,complex)`).

```
> fsolve(x^5+2*x+1,x,complex);
```

$$-.7018735689-.8796971979\,I\ -.7018735689+.8796971979\,I\ -.4863890359$$
$$.9450680868-.8545175144\,I\ .9450680868+.8545175144\,I$$

La représentation `RootOf` permet de calculer dans une extension algébrique des rationnels. Pour travailler dans le corps des racines du polynôme x^2+x+1, on écrit d'abord

```
> alias(j = RootOf(x^2+x+1));
```

$$I,\ j$$

La commande `alias` renvoie la séquence des *alias*, qui par défaut contient `I` défini comme `(-1)^(1/2)`. L'alias n'est qu'une abréviation qui donne un plus grand confort à l'utilisateur. Ensuite on utilise la procédure `evala` d'évaluation des nombres algébriques

```
> evala(1/(j+1));
```

$$-j$$

Maple ne réduit pas systématiquement les fractions rationnelles, contrairement à ce qu'il fait pour les nombres rationnels.

```
> f:=1/(x^2+x+1)+1/(x^2-1);
```

$$f:=\frac{1}{x^2+x+1}+\frac{1}{x^2-1}$$

La forme réduite s'obtient grâce à la commande `normal` :

```
> f:=normal(f);
```

$$f:=\frac{x(2x+1)}{(x^2+x+1)(x^2-1)}$$

Les commandes `denom(f)` et `numer(f)` renvoient le dénominateur et le numérateur de f. La décomposition en éléments simples sur le corps de base s'obtient en invoquant `convert/parfrac`

```
> convert(f,parfrac,x);
```

$$\frac{1}{2}\frac{1}{x-1}-\frac{1}{2}\frac{1}{x+1}+\frac{1}{x^2+x+1}$$

Pour obtenir la décomposition sur le corps des racines du dénominateur de f, il faut le factoriser complètement

```
> f:=factor(f,j);
```

$$f:=\frac{x(2x+1)}{(x+1+j)(x-j)(x+1)(x-1)}$$

```
> convert(f,parfrac,x);
```

$$-\frac{j+1}{(j+2)\,j\,(x+1+j)}+\frac{j}{(j^2-1)\,(x-j)}+\frac{1}{(2\,j+2)\,j\,(x+1)}-\frac{3}{(2\,j-2)\,(j+2)\,(x-1)}$$

On peut ensuite simplifier le résultat dans le corps $\mathbb{Q}(j)$ en appliquant `evala` à chacun des termes de cette dernière expression :

```
> map(evala,");
```

$$\frac{\frac{2\,j}{3}+1/3}{x+1+j}+\frac{-1/3-\frac{2\,j}{3}}{x-j}-\frac{1}{2\,x+2}+\frac{1}{2\,x-2}$$

3.4. Algèbre linéaire. Le *package* `linalg` donne accès à des commandes d'algèbre linéaire. Pour déclarer la matrice

$$A=\begin{pmatrix}1 & a & b\\ a & 1 & c\end{pmatrix},$$

on peut employer la forme longue

```
> A:=linalg[matrix](2,3,[1,a,b,a,1,c]);
```

$$A:=\begin{bmatrix}1 & a & b\\ a & 1 & c\end{bmatrix}$$

ou, si l'on compte faire une utilisation intensive du *package* `linalg`, la forme courte après avoir redirigé les noms avec `with`

```
> with(linalg):
> A:=matrix(2,3,[1,a,b,a,1,c]):
```

On peut même écrire

```
> A:=matrix([[1,a,b],[a,1,c]]):
```

En fait, une matrice est représentée par un tableau bidimensionnel et l'on aurait pu définir la matrice, sans avoir chargé les noms du *package* `linalg`, par

```
> A:=array(1..2,1..2,[[1,a,b],[a,1,c]]):
```

Pour déclarer une matrice $m\times n$ dont le terme d'indice (i,j) est $f(i,j)$, où f est une procédure, on utilise `matrix(m,n,f)`. Par exemple

```
> A:=matrix(3,3,(i,j)->i^(j-1));
```

$$A:=\begin{bmatrix}1 & 1 & 1\\ 1 & 2 & 4\\ 1 & 3 & 9\end{bmatrix}$$

Le tableau 4 fournit les principales commandes disponibles dans le *package* `linalg`. Les définitions précises seront obtenues en utilisant le *help*. Pour l'utilisation ponctuelle d'une procédure d'un package, on ne charge généralement pas les noms du *package* ce qui évite que certaines procédures en mode standard soient redéfinies (certaines procédures de package ont le même nom que des procédures standard), mais cela oblige à utiliser les formes longues des noms de procédure. Parallèlement au *package* MAPLE propose une procédure `evalm` qui permet d'effectuer des calculs disons syntaxiques sur les matrices. Il faut noter que le produit, qui est non commutatif, s'écrit avec `&*` et non avec `*`. L'instruction `evalm` force le calcul effectif des expressions sur les matrices, qui sinon sont vues comme des objets symboliques :

```
> A:=matrix(3,3, (i,j)->i*j ); B:=vandermonde([x,y,z]);
```

$$A:=\begin{bmatrix}1 & 2 & 3\\ 2 & 4 & 6\\ 3 & 6 & 9\end{bmatrix}$$

`matrix`	définition d'une matrice
`add(A,B)`	somme des matrices A et B
`add(A,B,lambda,mu)`	combinaison linéaire $\lambda A + \mu B$
`multiply(A,B)`	produit des matrices A et B
`rank(A)`	rang de la matrice A
`det(A)`	déterminant de A
`array(1..n,1..n,identity)`	matrice identique d'ordre n
`inverse(A)`	inverse de A
`transpose(A)`	matrice transposée de A
`nullspace(A)`	noyau de A
`charpoly(A,x)`	polynôme caractéristique de A
`eigenvals(A)`	valeurs propres de A
`eigenvects(A)`	vecteurs propres de A
`A+B`	somme des matrices A et B
`alpha*A`	produit de la matrice A par le scalaire α
`A &* B`	produit des matrices A et B
`A^k`	A^k
`&*()`	matrice identique
`A^(-1)`	A^{-1}

TABLEAU 4. Les procédures ou expressions courantes en algèbre linéaire. Les premières font partie du package `linalg` ; les secondes demandent l'utilisation d'`evalm`.

$$B := \begin{bmatrix} 1 & x & x^2 \\ 1 & y & y^2 \\ 1 & z & z^2 \end{bmatrix}$$

```
> C:=A&*B;
```

$$C := A\&*B$$

```
> evalm(C);
```

$$\begin{bmatrix} 6 & x+2\,y+3\,z & x^2+2\,y^2+3\,z^2 \\ 12 & 2\,x+4\,y+6\,z & 2\,x^2+4\,y^2+6\,z^2 \\ 18 & 3\,x+6\,y+9\,z & 3\,x^2+6\,y^2+9\,z^2 \end{bmatrix}$$

Il est possible de déclarer un vecteur comme une matrice colonne par `array(1..n,1..1)`. MAPLE possède une commande plus directe :

```
> v:=vector([x,y,z]);
```

$$v := [x \quad y \quad z]$$

Pour gagner de la place, les vecteurs sont écrits en ligne, mais il s'agit bien de vecteurs colonnes. Les opérations de base sont analogues à celles des matrices.

```
> A:=vandermonde([x,y,z]): evalm(A&*v);
```

$$[x+xy+x^2z \quad x+y^2+y^2z \quad x+zy+z^3]$$

La commande `transpose` sur les vecteurs s'applique mais, même avec `evalm`, MAPLE interdit de voir le résultat ; ceci est un choix des concepteurs à cause de la représentation des vecteurs colonnes en ligne ; pour qu'un vecteur soit représenté en colonne il faut le déclarer comme une matrice $n \times 1$.

Le produit scalaire hermitien s'obtient avec `dotprod` et le produit vectoriel s'obtient avec `crossprod`.

```
> v:=vector([1,I]): w:=vector([I,1]):
> dotprod(v,w);
```

$$0$$

```
> v:=vector([seq(x[i],i=1..3)]);w:=vector([seq(y[i],i=1..3)]);
```

$$v := [\ x_1 \quad x_2 \quad x_3\ ]$$

$$w := [\ y_1 \quad y_2 \quad y_3\ ]$$

```
> crossprod(v,w);
```

$$[\ x_2y_3 - x_3y_2 \quad x_3y_1 - x_1y_3 \quad x_1y_2 - x_2y_1\ ]$$

4. Objets Maple

Les notions rencontrées jusqu'ici permettent de résoudre des problèmes sans vraiment se préoccuper du fonctionnement de Maple. Cependant une réelle compréhension du système nécessite l'étude des objets manipulés d'un point de vue plus informatique. On y gagne une plus grande efficacité et une certaine sûreté dans le maniement du langage.

Le calcul formel rend effectifs les calculs de l'algèbre classique au travers d'algorithmes dont la théorie s'appuie sur les mathématiques mais dont la mise en place ressortit à l'informatique. Le passage de l'un à l'autre fait glisser de concepts sémantiques à des concepts syntaxiques : dans le premier cas on utilise des opérations définies sur des ensembles pourvues de certaines propriétés ; dans le second on utilise des symboles et des grammaires. Un logiciel de calcul formel, à la frontière de ces deux conceptions, utilise des expressions syntaxiquement correctes que l'utilisateur interprète comme des éléments d'un ensemble. Le paragraphe précédent a mis en valeur l'interprétation mathématique des termes et nous allons ici nous appesantir sur la syntaxe des termes, que l'on nommera plutôt expression. Ces expressions sont les éléments de base de Maple et nous retiendrons donc la règle suivante.

L'entité de base est l'expression.

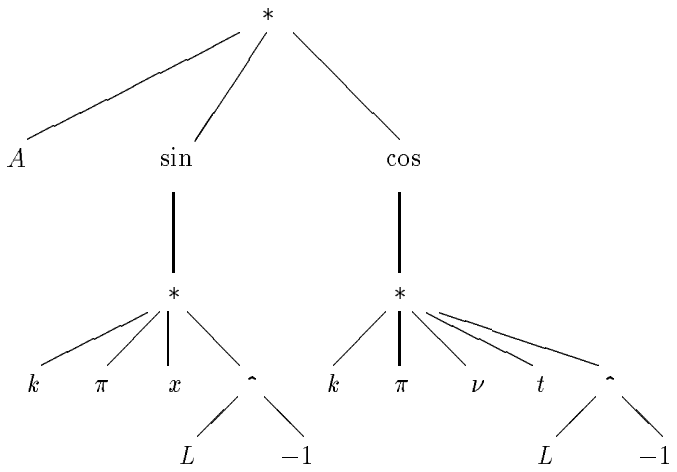
C'est pourquoi nous avons dit plus haut qu'une fonction est représentée par une expression. Dans ce cadre un polynôme est une expression de la forme $a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ comme on l'enseignait autrefois.

4.1. Structure. Un objet Maple est un arbre, au sens informatique du terme. Par exemple l'expression

```
> expr:=A*sin(k*Pi*x/L)*cos(k*Pi*nu*t/L);
```

$$expr := A \sin\left(\frac{k\pi\, x}{L}\right) \cos\left(\frac{k\pi\, \nu\, t}{L}\right)$$

correspond à l'arbre suivant.



On obtient le type de la racine de l'arbre par la commande **whattype** et la liste des opérandes, c'est-à-dire des sous-arbres attachés à la racine par **op**. Ici l'expression est un produit et possède trois opérandes.

```
> whattype(expr);
```

$$*$$

```
> op(expr);
```

$$A, \sin\left(\frac{k\pi\, x}{L}\right), \cos\left(\frac{k\pi\, \nu\, t}{L}\right)$$

Le nombre d'opérandes s'obtient par **nops** et on peut isoler chacune des opérandes en utilisant son rang et ainsi descendre dans l'arbre pour obtenir n'importe quelle sous-expression.

```
> nops(expr);
```

$$3$$

```
> op(2,expr);whattype(op(2,expr));
```

$$\sin\left(\frac{k\pi\, x}{L}\right)$$

$$function$$

```
> op(op(2,expr));op(");
```

$$\frac{k\pi x}{L}$$

$$k, \pi, x, \frac{1}{L}$$

Dans une session donnée l'ordre des opérandes est fixé, mais d'une session à l'autre l'ordre des opérandes peut changer ; on ne peut donc pas fonder un programme sur l'ordre des opérandes d'une expression.

Signalons que pour la plupart des expressions ont peut obtenir la racine comme l'opérande d'indice 0 (les exceptions sont les séries et les tables).

```
> op(0,expr);op(0,op(2,expr));
```

$$*$$

$$\sin$$

4.2. Types de base. Le *help* de **type** nous donne les différents types standard, parmi lesquels on trouve les types produit (*****), somme (**+**), tableau (**array**), booléen (**boolean**), nombre complexe (**complex**), équation (**equation**), nombre flottant (**float**), nombre entier (**integer**), matrice (**matrix**), polynôme (**polynom**), intervalle (**range**), nombre rationnel (**rational**), chaîne de caractères (**string**) ou série de Taylor (**taylor**). Nous n'allons pas décrire tous ces types, dont beaucoup ont un sens assez clair et auquel le *help* permet de s'initier. Nous nous contenterons d'insister sur les types table et procédure.

Le type table est d'une grande importance car les tableaux (*array*) sont des tables indexées par des entiers variant dans des intervalles et les matrices sont des tableaux dont les indices sont dans des intervalles d'entiers de la forme $[1, N]$. Quant aux procédures elles sont la base même de la programmation.

On peut créer une table comme suit.

```
> PhysConst:=table([c=2.9979250*10^8*m/s,e=1.6021917*10^(-19)*C,
N=6.022169*10^23*mol^(-1),u=1.660531*10^(-27)*kg,h=6.626196*10^(-34)*J*s]):
```

Une des difficultés que présentent les tables et les procédures est leur mécanisme d'évaluation. En effet l'évaluation d'une table ou d'une procédure produit son nom.

```
> PhysConst;
```

$$PhysConst$$

Il faut forcer l'évaluation par `eval` ou l'écriture par `print` pour afficher le contenu de la table.

```
> print(PhysConst);
```

$$\begin{aligned} PhysConst := \text{table}([\\ h &= .6626196\,10^{-33} Js \\ N &= .6022169\,10^{24} \frac{1}{mol} \\ e &= .16021917\,10^{-18} C \\ u &= .1660531\,10^{-26} kg \\ c &= .2997925000\,10^{9} \frac{m}{s} \\ ]) \end{aligned}$$

On peut constater ce problème en utilisant `whattype`.

```
> A:=array([[a,b],[c,d]]);
```

$$A := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> whattype(A)
```

$$string$$

```
> whattype(eval(A));
```

$$array$$

Ici il faut forcer l'évaluation de la matrice A pour voir qu'il s'agit d'un tableau.

La commande `op` ne fonctionne pas sur les tables et sur les procédures comme sur les autres objets. Nous avons écrit dans un fichier une petite procédure qui fournit les entiers rencontrés à partir d'un entier donné en appliquant l'itération de la conjecture $3x+1$. Rappelons que cette conjecture dit que partant d'un entier naturel non nul quelconque on arrive à 1 en un nombre fini de pas si l'on applique les règles suivantes : un entier pair est divisé par 2 ; un entier impair est multiplié par 3 puis augmenté de 1. Nous faisons l'hypothèse que cette conjecture est correcte, le test d'arrêt étant la rencontre de 1. Nous lisons le fichier puis nous regardons le corps de la procédure et ses différentes opérandes.

```
> read `Maple/conjecture3x+1.mpl`;
> print(optim);

proc(x)
options remember;
    if x = 1 then 1
    elif type(x,even) then x,optim(1/2*x)
    else x,optim(3*x+1)
    fi
end

> seq([i,op(i,eval(optim))],i=0..6);
```

$$[0, \mathit{procedure}], [1, x], [2], [3, \mathit{remember}], [4],$$

$$[5, \mathit{optim(x)\ renvoie\ la\ suite\ des\ entiers\ rencontres\ a\ partir\ de\ x}], [6]$$

L'application de `op` ou de `eval` fournit le corps de la procédure et les différents champs sont, respectivement à partir de l'indice 0, le type, les paramètres, les variables locales, les options, la table de *remember*, la description et les variables globales. Ici il n'y a ni variables locales, ni variables globales ; l'unique paramètre est x ; il y a une description, ce qui est plutôt rare (cette possibilité n'est pas utilisée pour les procédures de la bibliothèque standard), et une table de *remember* qui est vide pour l'instant. Une simple évaluation suffit à la nourrir.

```
> optim(10);
```

$$10, 5, 16, 8, 4, 2, 1$$

```
> seq([i,op(i,eval(optim))],i=0..6);
```

$$[0, \mathit{procedure}], [1, x], [2], [3, \mathit{remember}], [4, \mathrm{table}([$$
$$1 = 1$$
$$2 = (2, 1)$$
$$4 = (4, 2, 1)$$
$$5 = (5, 16, 8, 4, 2, 1)$$
$$8 = (8, 4, 2, 1)$$
$$10 = (10, 5, 16, 8, 4, 2, 1)$$
$$16 = (16, 8, 4, 2, 1)$$
$$])] [5, \mathit{optim(x)\ renvoie\ la\ suite\ des\ entiers\ rencontres\ a\ partir\ de\ x}], [6]$$

Ensuite les tables et les procédures peuvent amener des créations implicites. Par exemple le fait d'écrire

```
> t[Marignan]:=1515:
```

crée une table de nom `t`.

```
> eval(t);
```

$$\mathrm{table}([$$
$$\mathit{Marignan} = 1515$$
$$])$$

De la même façon la commande

```
> f(Waterloo):=1814;
```

crée une procédure de nom `f`, dont la table de *remember*

```
> op(4,eval(f));
```

$$\mathrm{table}([$$
$$\mathit{Waterloo} = 1814$$
$$])$$

ne contient pour l'instant que la valeur qui vient d'être définie et qui ne fait rien pour un autre argument que `Waterloo`.

```
> f(x);
```

$$f(x)$$

Enfin les tables sont les seuls objets qui peuvent être modifiés. Ayant défini l'expression f par

```
> f:=erf((x-1)/sqrt(2)):
```

la suite d'instructions

```
> diff(f,x);
```

$$\frac{e^{-1/2(x-1)^2}\sqrt{2}}{\sqrt{\pi}}$$

```
> subs(x=2,f);
```

$$\mathrm{erf}\left(\frac{\sqrt{2}}{2}\right)$$

ne modifie pas l'expression f,

```
> f;
```

$$\mathrm{erf}\left(\frac{1}{2}(x-1)\sqrt{2}\right).$$

Évidemment on peut affecter au nom f un nouvel objet, mais l'expression qui avait pour nom f n'a pas été modifiée. Par contre une instruction comme `toto[titi]:=tutu` modifie la table de nom *toto*. On se souviendra donc de la règle,

Les tables sont les seuls objets qui peuvent être modifiés.

De plus une affectation comme

```
> NewPhysConst:=PhysConst:
```

ne crée pas un nouvel objet mais fait que les deux noms `PhysConst` et `NewPhysConst` pointent sur le même objet. Pour le voir, modifions l'objet de nom $NewPhysConst$.

```
> NewPhysConst[re]:=2.817939*10^(-15)*m;
```

$$NewPhysCons_{re} := .2817939\,10^{-14}m$$

```
> print(PhysConst);
```

$$\begin{aligned}
&PhysConst := \mathrm{table}([\\
&\quad h = .6626196\,10^{-33}Js\\
&\quad N = .6022169\,10^{24}\frac{1}{mol}\\
&\quad e = .16021917\,10^{-18}C\\
&\quad re = .2817939\,10^{-14}m\\
&\quad u = .1660531\,10^{-26}kg\\
&\quad c = .2997925000\,10^{9}\frac{m}{s}\\
&\quad])
\end{aligned}$$

La modification apportée à une table a aussi modifié l'autre. Pour éviter ceci il faut utiliser la commande `copy`, qui crée une nouvelle table.

4.3. Créations d'expressions. Un objet MAPLE qui n'est pas une table ne peut être modifié et l'application d'une commande à un objet crée un nouvel objet. Passons en revue quelques commandes usuelles qui permettent de créer de nouvelles expressions à partir d'expressions existantes.

La commande `convert` modifie essentiellement la racine de l'arbre associé à une expression. Utilisons `convert` pour additionner les éléments d'une liste.

```
> L:=[seq(x[(i*(i+1)/2)],i=1..10)];
```

$$L := [x_1, x_3, x_6, x_{10}, x_{15}, x_{21}, x_{28}, x_{36}, x_{45}, x_{55}]$$

```
> op(0,L);op(L);
```

$$list$$

$$x_1, x_3, x_6, x_{10}, x_{15}, x_{21}, x_{28}, x_{36}, x_{45}, x_{55}$$

```
> S:=convert(L,`+`);
```

$$x_1 + x_3 + x_6 + x_{10} + x_{15} + x_{21} + x_{28} + x_{36} + x_{45} + x_{55}$$

```
> op(0,S);op(S);
```

$$+$$

$$x_1, x_3, x_6, x_{10}, x_{15}, x_{21}, x_{28}, x_{36}, x_{45}, x_{55}$$

On utilise fréquemment `convert` pour calculer une somme ou un produit (pour chercher une formule sommatoire on utilise `sum`). La transformation effectuée par `convert` n'est pas toujours aussi simple que dans le cas précédent. Une série de Taylor est une expression dont la racine est étiquetée par le nom de la variable et dont les opérandes sont alternativement les coefficients et les exposants de la série.

```
> S:=series(x/(exp(x)-1),x);
```

$$S := 1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{1}{720}x^4 + O\left(x^5\right)$$

```
> op(0,S);op(S);
```

$$x$$

$$1, 0, \frac{-1}{2}, 1, \frac{1}{12}, 2, \frac{-1}{720}, 4, O(1), 5$$

En utilisant `convert/polynom` on obtient la partie régulière du développement.

```
> P:=convert(S,polynom);
```

$$P := 1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{1}{720}x^4$$

```
> op(0,P); op(P);
```

$$+$$

$$1, -\frac{1}{2}x, \frac{1}{12}x^2, -\frac{1}{720}x^4$$

La commande `subsop` permet de modifier l'une des sous-expressions renvoyées par `op`. Elle est par exemple utilisée pour vider la table de *remember* d'une procédure, disons `f`, par la commande

```
> f:= subsop(4=NULL,eval(f)):
```

Ceci peut être utile si on modifie le contexte dans lequel est utilisée une procédure. Par exemple, si une procédure renvoie des valeurs flottantes, les résultats dépendent de la valeur de la variable d'environnement `Digits` ; si on augmente cette valeur il faut vider la table de *remember* de la procédure pour que les résultats soient effectivement recalculés.

La commande `subs` permet de substituer à une sous-expression une autre expression. D'après le principe qui veut que l'on n'affecte pas une variable mathématique, `subs` est la voie normale pour évaluer une expression pour un jeu de valeurs des variables mathématiques qui y figurent. Voici par exemple les valeurs prises par le polynôme P en les points $k/10$ pour k allant de 0 à 5.

```
> seq(evalf(subs(x=k/10,P)),k=0..5);
```

$$1., .9508331944, .9033311111, .8574887500, .8132977778, .7707465278$$

Reprenons la série de Taylor S vue un peu plus haut.

```
> subs(x=sqrt(y),S);
```

$$1 - \frac{1}{2}\sqrt{y} + \frac{1}{12}y - \frac{1}{720}y^2 + O\left(y^{5/2}\right)$$

```
> subs(x^2=y,S);
```

$$1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{1}{720}x^4 + O\left(x^5\right)$$

La première commande fournit un développement de $\sqrt{y}/(e^{\sqrt{y}} - 1)$ et la seconde renvoie S elle-même car x^2 n'est pas une sous-expression de S, comme on l'a vu en tapant `op(S)`.

L'application de `subs` ne semble pas suivie d'une évaluation, ce qui peut être troublant. On le voit ci-après où l'on vérifie que e^t est solution de $z'' - 2z' + z = 0$.

```
> equ:=diff(z(t),t$2)-2*diff(z(t),t)+z(t):
> subs(z(t)=exp(t),equ);
```

$$\left(\frac{\partial^2}{\partial t^2}e^t\right) - 2\left(\frac{\partial}{\partial t}e^t\right) + e^t$$

```
> ";
```

$$0$$

Nous reviendrons bientôt sur ce point.

La commande `map` permet d'appliquer une procédure à toutes les opérandes d'une expression. Ici on arrange les termes d'un développement limité.

```
> series(1/(1-a*t)/(1-b*t),t);
```

$$1 + (a+b)\,t + \left(a^2 - (-a-b)\,b\right)t^2 + \left(a^3 - \left(-a^2 - ba - b^2\right)b\right)t^3 + \left(a^4 - \left(-a^3 - ba^2 - b^2a - b^3\right)b\right)t^4 + \left(a^5 - \left(-a^4 - ba^3 - b^2a^2 - b^3a - b^4\right)b\right)t^5 + O\left(t^6\right)$$

```
> map(normal,");
```

$$1 + (a+b)\,t + \left(a^2 + ba + b^2\right)t^2 + \left(a^3 + ba^2 + b^2a + b^3\right)t^3 + \left(a^4 + ba^3 + b^2a^2 + b^3a + b^4\right)t^4 + \left(a^5 + ba^4 + b^2a^3 + b^3a^2 + b^4a + b^5\right)t^5 + O\left(t^6\right)$$

Enfin la commande `select` sélectionne les opérandes d'une expression suivant un critère exprimé par une procédure à valeurs booléennes. On peut par exemple obtenir la partie homogène de degré 2 du polynôme $(1 + x + y^2)^4$ de la façon suivante.

```
> Q:=expand((1+x+y^2)^4);
```

$$Q := 1 + 4\,x + 4\,y^6 + 4\,y^2 + 6\,y^4 + 12\,xy^4 + 12\,x^2y^2 + 12\,xy^2 + 6\,x^2 + 4\,x^3 + x^4 + 4\,x^3y^2 + 6\,x^2y^4 + 4\,xy^6 + y^8$$

```
> select(u->evalb(degree(u)=2),Q);
```

$$4\,y^2 + 6\,x^2$$

5. Simplification et évaluation

La simplification et l'évaluation sont deux mécanismes de base, puisque Maple, en utilisation interactive, lit une expression, l'analyse syntaxiquement, la simplifie et l'évalue puis renvoie la valeur obtenue.

5.1. Simplification. La notion de simplification est au cœur du calcul formel mais n'a de sens que dans un contexte donné. Par exemple l'expression développée d'un polynôme est adéquate si l'on s'intéresse à ses coefficients mais la forme factorisée convient beaucoup mieux à l'étude de ses racines. Suivant les besoins on voudra donc passer d'une forme à une autre et nous allons décrire les différents outils que propose Maple.

Remarquons d'abord que Maple procède à une simplification dite automatique indépendamment de l'utilisateur. On le voit clairement même si on essaye de bloquer l'évaluation en entourant les expressions de *quotes*.

```
> 'x+y-x';
```

$$y$$

```
> 'x*y*x*3';
```

$$3x^2y$$

À l'occasion ceci peut être gênant. Supposons que l'on veuille étudier la somme de deux variables aléatoires à valeurs dans $\{0,1\}$.

```
> coin:=rand(2): somme_var:=proc() coin()+coin() end:
```

Avec cette définition, la simplification automatique fait que `somme_var()` se comporte comme le double de `coin()` alors que l'on pourrait espérer qu'il apparaisse comme la somme de deux tirages aléatoires successifs.

```
> seq(somme_var(),i=1..10);
```

$$0, 0, 2, 2, 2, 0, 0, 2, 2, 0$$

```
>'coin()+coin()';
```

$$2\,\mathrm{coin}()$$

Ensuite le système applique des règles d'évaluation qui traduisent certaines propriétés attendues des fonctions usuelles.

```
> sin(0),sin(Pi),sin(-x),diff(sin(x),x);
```

$$0, 0, -\sin(x), \cos(x)$$

D'autres simplifications doivent être demandées explicitement par l'utilisateur grâce aux procédures `normal`, `expand`, `combine`, `simplify`, `convert`. Insistons sur le fait qu'une utilisation aveugle ne produit généralement rien de bon et qu'il faut donner des commandes précises en passant en second paramètre une option comme `trig`, `exp`, `ln` ou `power`. La fonction principale d'`expand` est d'appliquer les règles de distributivité ; elle permet aussi d'appliquer certains développements pour les fonctions usuelles ; `combine(`*expr*`,trig)` permet de linéariser les fonctions trigonométriques ; `convert(`*expr*`,exp)` permet d'appliquer les formules d'Euler.

```
> expand(cos(5*x));
```

$$16\cos(x)^5 - 20\cos(x)^3 + 5\cos(x)$$

```
> combine(cos(x)^5);
```

$$\cos(x)^5$$

```
> combine(cos(x)^5,trig);
```

$$\frac{1}{16}\cos(5\,x)+\frac{5}{16}\cos(3\,x)+\frac{5}{8}\cos(x)$$

```
> convert(",exp);
```

$$\frac{1}{32}e^{(5Ix)}+\frac{1}{32}\frac{1}{e^{(5Ix)}}+\frac{5}{32}e^{(3Ix)}+\frac{5}{32}\frac{1}{e^{(3Ix)}}+\frac{5}{16}e^{(Ix)}+\frac{5}{16}\frac{1}{e^{(Ix)}}$$

```
> op(1,");
```

$$\frac{1}{32}e^{(5Ix)}$$

```
> convert(",trig);
```

$$\frac{1}{32}\cos(5x)+\frac{1}{32}I\sin(5x)$$

La simplification est un problème délicat et nous ne pouvons que vous encourager à lire la feuille d'aide de *simplify* et celles qui lui sont associées. Le *help* de `simplify/siderels` montre par exemple comment utiliser les bases de Gröbner à travers l'instruction suivante.

```
> simplify(cos(x)^8+sin(x)^8,{cos(x)^2+sin(x)^2=1});
```

$$2\sin(x)^8-4\sin(x)^6+6\sin(x)^4-4\sin(x)^2+1$$

5.2. Évaluation. L'évaluation est récursive et consiste à remplacer chaque variable qui apparaît dans une expression par sa valeur et à appliquer chaque fonction à ses opérandes. Lors de cette évaluation, une variable qui n'est pas affectée ou est affectée à une table ou une procédure a pour valeur son nom ; chaque opérande d'une fonction est évaluée avant que celle-ci ne soit appliquée (à quelques exceptions près comme `evalf` ou `time`).

Dans certains cas ceci peut être légèrement déroutant. On a par exemple

```
> subs(x=0,sin(x));
```

$$\sin(0)$$

et on voit que l'évaluation ne produit pas ce que l'on pourrait attendre. En effet dans l'évaluation de `subs(x=0,sin(x))` Maple évalue d'abord les deux arguments ce qui lui fournit l'équation $x=0$ et l'expression $\sin(x)$ puis il opère la substitution ce qui fournit l'expression $\sin(0)$; il y a donc bien évaluation. Si l'on veut obtenir la valeur attendue de l'expression avant de poursuivre, il faut utiliser `eval`.

Nous avons dit que l'utilisation des *quotes* bloquait l'évaluation. L'évaluation de *'expr'* est simplement *expr*. Il y a donc en fait évaluation. On le voit bien sur l'exemple suivant où chaque évaluation enlève une paire de *quotes*.

```
> ''exp(0)'';
```

$$'e^{0}'$$

```
> ";
```

$$e^{0}$$

```
> ";
```

$$1$$

5.3. Formes inertes. Dans certaines situations il est utile de considérer une expression sans l'évaluer. Par exemple on peut vouloir opérer un changement de variables ou une intégration par parties sur une intégrale sans pour autant laisser MAPLE la calculer. Dans ce but on dispose de formes inertes comme `Int` pour les intégrales ou `RootOf` pour les racines d'une équation. Une telle procédure ne fait rien mais on peut appliquer certaines fonctions à l'expression qu'elle définit. Cherchons par exemple les valeurs propres de la matrice symétrique suivante sous forme numérique.

```
> A:=array(symmetric,[seq([seq(exp(i-j),j=1..6)],i=1..6)]);
```

$$\begin{bmatrix} 1 & e^1 & e^2 & e^3 & e^4 & e^5 \\ e^1 & 1 & e^1 & e^2 & e^3 & e^4 \\ e^2 & e^1 & 1 & e^1 & e^2 & e^3 \\ e^3 & e^2 & e^1 & 1 & e^1 & e^2 \\ e^4 & e^3 & e^2 & e^1 & 1 & e^1 \\ e^5 & e^4 & e^3 & e^2 & e^1 & 1 \end{bmatrix}$$

Si nous utilisons `linalg[eigenvals]` le système essaye d'obtenir les valeurs propres sous forme exacte. Dans ce cas particulier l'équation caractéristique est résoluble par radicaux et au bout d'un temps assez long le système donne une expression énorme, que l'on peut d'ailleurs simplifier mais qui reste tout de même trop grosse pour être écrite ici. On se résout alors à appliquer `evalf`. Les commandes suivantes sont bien plus efficaces et fournissent le résultat numérique en moins d'une seconde.

```
> Eigenvals(A);
```

$$Eigenvals(A)$$

```
> evalf(");
```

$$[-166.8710692, -1.434372126, -0.8533449711, -0.6003721083, -0.4925633688, 176.2517219]$$

6. PROGRAMMATION

La programmation en MAPLE a déjà été présentée à la section 2 et ce complément vise surtout à insister sur la propreté des programmes. En effet le langage MAPLE permet tous les errements s'il est mal employé et il faut donc s'imposer une discipline en écrivant des programmes complètement spécifiés et clairement documentés.

6.1. Vérification de type. Une procédure s'applique à des arguments qui doivent être d'un certain type. Depuis la version V.2, MAPLE fournit un mécanisme de vérification automatique des types qui évite une suite de tests en début de procédure. On peut utiliser les types de base ou des types composés (cf. le *help* de `type[structured]`). La syntaxe d'utilisation est la suivante

$$\texttt{proc}(\textit{paramètre 1} : \textit{type 1}, \ldots, \textit{paramètre n} : \textit{type n})$$

Supposons que vous définissiez une procédure qui extrait d'une liste de nombres les éléments strictement positifs. Vous pouvez procéder ainsi.

```
> extractpos:=proc(L:list(numeric)) select(x->evalb(x>0),L) end:
```

On pourrait aussi utiliser la commande `select(type,L,positive)`. Testons cette procédure.

```
> extractpos([2,-3,355/113,exp(1000.),sin(10.),ln(2.)]);
```

$$\left[2, \frac{355}{113}, .1970071114\, 10^{435}, .6931471806\right]$$

```
> extractpos([2,-3,Pi]);
Error, extractpos expects its 1st argument, L, to be of type list(numeric),
but received [2, -3, Pi]
```

La constante π n'est pas de type `numeric` (c'est-à-dire n'est ni un entier, ni un rationnel ni un flottant) et ceci a provoqué une erreur.

Des situations plus complexes sont imaginables comme la procédure suivante qui prend en entrée une matrice à coefficients des polynômes à coefficients entiers, un entier et une liste composée de noms ou d'entiers.

```
> toto:=proc(A:matrix(polynom(integer)),n:integer,L:list({name,integer}))
  print(`coucou`) end:
> toto(array([[3+x+y,-3+x^2],[2-y-z,0]]),3,[2,x,-3,Pi]);
```

coucou

Il faut cependant réfléchir à la structure des objets utilisés. Par exemple le type somme d'une exponentielle et d'un polynôme, `&+(exp(name),polynom)`, est syntaxiquement correct mais n'a pas de sens. En effet si on le teste sur $\exp(x)+3+x+x^2$ on se rend compte que Maple voit ceci comme la somme de quatre termes et non de $\exp(x)$ et $3+x+x^2$; il suffit de penser à l'arbre sous-jacent pour s'en rendre compte.

Une autre méthode pour résoudre ce problème de vérification de type consiste à définir un type procédural ; la syntaxe est expliquée dans le *help* de `type/argcheck` (avec un certain manque de clarté). Un exemple fera comprendre comment on procède.

L'étude des équations différentielles linéaires à coefficients constants amène à s'intéresser aux polynômes à coefficients réels dont toutes les racines ont une partie réelle strictement négative. Hurwitz a énoncé un critère [6] qui associe au polynôme

$$a_0 x^d + a_1 x^{d-1} + \cdots + a_d$$

les déterminants

$$\Delta_k = \begin{vmatrix} a_1 & a_3 & a_5 & \dots & a_{2k-1} \\ a_0 & a_2 & a_4 & \dots & a_{2k-2} \\ 0 & a_1 & a_3 & \dots & a_{2k-3} \\ & \dots & & \dots & \\ & \dots & & \dots & a_k \end{vmatrix}.$$

et qui s'exprime comme suit :

> *Une condition nécessaire et suffisante pour que l'équation*
>
> $$a_0 x^d + a_1 x^{d-1} + \cdots + a_d = 0$$
>
> *à coefficients réels, a_0 étant strictement positif, n'ait que des racines dont la partie réelle est strictement négative est que tous les déterminants*
>
> $$\Delta_1, \Delta_2, \dots, \Delta_d$$
>
> *soient strictement positifs.*

Sans discuter l'intérêt de ce résultat, nous voulons écrire une procédure Maple, `hurwitz`, qui le mette en pratique. Elle doit prendre en entrée un polynôme à coefficients réels, en une variable et de degré au moins égal à 1 et renvoyer un booléen. Nous définissons d'abord un type `hurwitzarg`, qui correspond aux polynômes à coefficients réels, en une variable, de degré strictement positif.

```
`type/hurwitzarg`:=proc(P)
  option remember;
  evalb( type(evalf(P),polynom(numeric)) # P doit etre un polynome a coefficients reels
    and nops(indets(P))=1)               # en une indeterminee non constant
end:
```

Le `evalf` est là pour autoriser des coefficients comme `exp(1)` ou `Pi` qui ne sont pas de type `numeric`. Ensuite nous employons ce type dans la procédure `hurwitz` pour vérifier que l'argument est valide.

```
hurwitz:=proc(P:hurwitzarg)
  local p,x,H,d,i,k;
  option remember;
  p:=expand(p);        # on normalise
  x:=op(indets(p));    # en appelant x la variable
  d:=degree(p,x);      # et en rendant le coefficient dominant positif
  if evalf(coeff(p,x,d))<0 then p:=-p fi;
  H:=array([[coeff(p,x,d-1)]]); # on construit la matrice [a_1]
  for i to d-1 do      # on mouline
# a chaque pas on teste si le determinant est strictement positif
# si ce n'est pas le cas on arrete tout et on renvoie false
    if not(evalf(linalg[det](H))>0) then RETURN(false) fi;
# on prepare le cran suivant en bordant la matrice
    H:=linalg[augment](
        linalg[stack](
         H,array([[seq(coeff(p,x,d-(i-1+2*k)),k=(-i+1)..0)]])
                    ),
          array([seq([coeff(p,x,d-(i+1-k))],k=-i..0)])
                     );
  od;
# au dernier cran il suffit de regarder le signe de a_d
  if not(evalf(coeff(p,x,0))>0) then RETURN(false) fi
# si on arrive jusqu'ici, c'est que le critere est satisfait
  true;
end:
```

Le caractère dièse `#` permet d'écrire des commentaires.

6.2. Aide à la programmation. L'utilisateur dispose de plusieurs outils dans l'écriture de ses programmes. Sous Unix, l'utilitaire Mint permet de vérifier la syntaxe. Pour cela on écrit dans un fichier les procédures voulues, puis on applique Mint à ce fichier. Le petit programme précédent écrit dans le fichier `hurwitz.mpl` n'amène aucun commentaire, car il est syntaxiquement correct.

```
mint hurwitz.mpl
    |\^/|     Maple V Release 2 Diagnostic Program
._|\|   |/|_. Copyright (c) 1981-1992 by the University of Waterloo.
 \  MINT   /  All rights reserved. Maple is a registered trademark of
 <____ ____>  Waterloo Maple Software.
      |
```

D'autre part on peut vérifier la sémantique en utilisant la variable `printlevel`. Par défaut sa valeur est 1 et plus on l'augmente plus on voit de détails d'exécution. Pour chercher une erreur on emploie souvent la valeur 100.

```
> printlevel:=10:
> hurwitz(2+3*z+4*z^2+z^3+z^4);
{--> enter hurwitz, args = 2+3*z+4*z^2+z^3+z^4
```

$$x := z$$

$$d := 4$$

$$p := 2 + 3z + 4z^2 + z^3 + z^4$$

```
{--> enter index/FillInitVals, args = MATRIX([[?[1,1]]]), [[1]]
```

$$T := \mathrm{array}(1..1, 1..1, [])$$

```
<-- exit index/FillInitVals (now in hurwitz) = }
```

$$H := [1]$$

$$H := \begin{bmatrix} 1 & 3 \\ 1 & 4 \end{bmatrix}$$

$$H := \begin{bmatrix} 1 & 3 & 0 \\ 1 & 4 & 2 \\ 0 & 1 & 3 \end{bmatrix}$$

$$H := \begin{bmatrix} 1 & 3 & 0 & 0 \\ 1 & 4 & 2 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 1 & 4 & 2 \end{bmatrix}$$

```
<-- exit hurwitz (now at top level) = true}
```

true

On peut aussi utiliser la procédure `trace`. D'autre part il peut être intéressant de regarder comment sont écrites les procédures de la bibliothèque (*library*) Maple fournies avec le système. Hormis celles qui sont dans le noyau (elles ont l'option *builtin*), elles ont toutes un code accessible en mettant la variable `verboseproc` à la valeur 2.

```
> interface(verboseproc=2):
```

Il suffit alors d'appliquer `print` ou `eval` à la procédure choisie pour lire son code.

7. Musculation

Voici quelques thèmes à traiter en Maple. Ils sont indépendants les uns des autres et essentiellement tournés vers le calcul formel.

7.1. Algorithme boustrophédon. Nous générons une pyramide d'entiers en procédant comme suit. On affecte au sommet de la pyramide la valeur 1 (ceci constitue la ligne d'indice 0 de la pyramide). On affecte ensuite les lignes les unes après les autres, en procédant comme le bœuf qui tire la charrue :

- on parcourt la ligne de gauche à droite si son indice est impair, de droite à gauche si son indice est pair ;
- au sein d'une même ligne, on écrit les nombres les uns après les autres en commençant par 0, puis en rajoutant à chaque fois la valeur du nombre de la ligne précédente dont la position verticale précède (dans le sens de parcours de la ligne courante) juste celle du nombre à écrire.

Les cinq premières lignes de la pyramide sont les suivantes

ligne 0				1				
ligne 1			0		1			
ligne 2		1		1		0		
ligne 3	0		1		2		2	
ligne 4	5		5		4		2	0

Pour tout n, appelons a_n et b_n les nombres se trouvant respectivement à gauche et à droite de la ligne d'indice n. On peut démontrer [2, 8] que

$$\sec(x) = \sum_{n=0}^{+\infty} \frac{a_n}{n!} \quad \text{et} \quad \tan(x) = \sum_{n=1}^{+\infty} \frac{b_n}{n!}.$$

Utilisez cette technique pour écrire deux procédures qui calculent les développements limités de sécante et de tangente à un certain ordre. On attend un résultat de la forme suivante.

```
> secseries(x,8);
```

$$1 + \frac{1}{2}x^2 + \frac{5}{24}x^4 + \frac{61}{720}x^6 + O(x^8).$$

7.2. Développement en série d'une fonction implicite. On se donne une fonction de deux variables F indéfiniment dérivable, nulle en $(0,0)$ et telle que sa dérivée partielle par rapport à sa seconde variable ne soit pas nulle en $(0,0)$. Le théorème des fonctions implicites assure l'existence et l'unicité d'une fonction régulière ϕ, définie sur un voisinage de 0, nulle en 0 et telle que

$$F(x, \phi(x)) = 0$$

sur un voisinage de 0. Le développement en série de ϕ au voisinage de 0 peut s'obtenir grâce à l'algorithme de Newton. On démontre [4, p. 510] que si $z_0(x) = 0$ et si $z_{n+1}(x)$ est la troncature jusqu'à $O(x^{2^{n+1}})$ du développement en série de

$$z_n(x) - \frac{F(x, z_n(x))}{\frac{\partial F}{\partial y}(x, z_n(x))},$$

alors $\phi(x) = z_n(x) + O(x^{2^n})$.

En utilisant cet algorithme, écrire une procédure MAPLE qui prend en entrée

- une expression définissant une fonction de deux variables $F(x, y)$;
- les deux variables x et y ;
- un entier n,

et qui renvoie le développement en série de la fonction implicite ϕ jusqu'à l'ordre n dans le cas régulier, $F(0,0) = 0$ et $\partial F/\partial y(0,0) \neq 0$.

On pourra tester la procédure avec $F(x, y) = e^y - 1 - x$ puis avec $F(x, y) = e^x - 1 + y\cos x + y^2 \sin x$.

7.3. Suite de Sturm. Étant donné un polynôme réel P, on définit sa *suite de Sturm* (P_n) par

$$P_0(x) = P(x), \quad P_1(x) = P'(x), \qquad P_i(x) = -\operatorname{reste}(P_{i-2}(x), P_{i-1}(x)) \quad \text{pour} \quad i \geq 2,$$

où $\operatorname{reste}(F, G)$ désigne le reste de la division euclidienne de F par G. On s'arrête au rang k lorsque P_k est un polynôme constant. Pour tout nombre réel x, on note $V(x)$ le nombre de changements de signe dans la suite des signes de $P_0(x), P_1(x), \ldots, P_k(x)$. Précisons qu'on ne compte pas les zéros ; par exemple, le nombre de changements de signe de la suite de signes $0, +, -, -, 0, +, +, -$ est égal à 3.

Le théorème de Sturm [3, Vol. 1, page 444] dit que le nombre de racines réelles de $P(x)$ se trouvant dans l'intervalle $]a, b]$ est égal à $V(a) - V(b)$ sous l'hypothèse que $P(x)$ est sans carré, c'est-à-dire que toutes ses racines sont simples.

Écrire une procédure `mysturm` qui prend en entrée un polynôme rationnel sans carré P et deux rationnels a et b, et qui renvoie le nombre de racines de P dans l'intervalle $]a, b]$. On pourra comparer les résultats obtenus avec ceux de la procédure `sturm` (pour l'employer il faut commencer par un `readlib(sturm)`).

On peut ensuite utiliser ceci pour localiser les racines réelles d'un polynôme. En procédant par dichotomie à partir de la procédure `mysturm`, écrire une procédure récursive `real_zeroes` qui prend en entrée

- un polynôme rationnel sans carré P et sa variable x ;
- un paramètre d'erreur $\varepsilon > 0$

et renvoie un ensemble de couples (I, ν) constitués d'un intervalle I et d'un entier strictement positif ν ; les intervalles I ont une longueur strictement inférieure à ε ; leur réunion recouvre l'ensemble des racines réelles de P ; le nombre de racines de P dans I est ν.

Pour démarrer, il faut un intervalle contenant toutes les racines réelles de P. On pourra utiliser le résultat suivant [3, Vol. 1, page 457] :

Toutes les racines α du polynôme $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ vérifient

$$|\alpha| \leq 2 \max\left(\left|\frac{a_{n-1}}{a_n}\right|, \left|\frac{a_{n-2}}{a_n}\right|^{1/2}, \left|\frac{a_{n-3}}{a_n}\right|^{1/3}, \ldots, \left|\frac{a_0}{a_n}\right|^{1/n} \right).$$

Voici un exemple d'application avec un polynôme de Tchebycheff.

```
> P:=sort(2*expand(subs(x=(x-1)/2,orthopoly[T](7,x))));
```

$$P := x^7 - 7x^6 + 15x^5 - 21x^3 + 7x^2 + 7x - 1$$

```
> real_zeroes(P,1);
```

$$\left\{ \left[\frac{7}{8}..\frac{7}{4}, 1\right], \left[0..\frac{7}{8}, 1\right], \left[\frac{7}{4}..\frac{21}{8}, 2\right], \left[\frac{21}{8}..\frac{7}{2}, 1\right], \left[\frac{-7}{4}..\frac{-7}{8}, 1\right], \left[\frac{-7}{8}..0, 1\right] \right\}$$

7.4. Séries rationnelles. Les séries rationnelles sont les séries

$$S(x) = \sum_{n \geq 0} u_n x^n,$$

dont les coefficients satisfont une relation de récurrence à coefficients constants,

$$u_n = a_1 u_{n-1} + a_2 u_{n-2} + \cdots + a_d u_{n-d}.$$

Plus proprement nous considérons des séries formelles à coefficients dans un corps, en pratique le corps des rationnels et aussi sa clôture algébrique par le simple fait de manipuler des racines de polynômes. On pourrait tout aussi bien dans ce cas parler de séries entières.

Il existe de nombreuses façons de caractériser les séries rationnelles [5, chap. 3]. On peut déclarer qu'une série est rationnelle si elle est égale au développement en série formelle d'une fraction rationnelle qui n'admet pas 0 pour pôle. Ou encore que ses coefficients satisfont une relation de récurrence comme on l'a déjà indiqué. Un énoncé équivalent s'appuie sur la considération de l'opérateur de translation qui transforme la série $\sum_{n\geq 0} u_n x^n$ en la série $\sum_{n\geq 0} u_{n+1} x^n$; la série est rationnelle si et seulement si le sous-espace engendré par la série sous l'action de cet opérateur est de dimension finie.

On peut aussi définir une telle série par une représentation linéaire L, Q, C dans laquelle L est une matrice ligne $1 \times N$, Q est une matrice carré $N \times N$ et C est une matrice colonne $N \times 1$. Une telle représentation ne fait que traduire l'action de l'opérateur de translation dans le sous-espace associé à la série, moyennant l'usage d'une base de ce sous-espace. Plus précisément la série $F(x)$ est définie par

$$F(x) = \sum_{n \geq 0} LQ^n C x^n$$

et on peut montrer qu'elle est donnée par

$$\begin{aligned}(1 - a_1 x - a_2 x^2 - \cdots - a_\ell x^\ell) F(x) = (1 - a_1 x - a_2 x^2 - \cdots a_{\ell-1} x^{\ell-1}) LC \\ + x(1 - a_1 x - \cdots a_{\ell-2} x^{\ell-2}) LQC + \cdots + x^{\ell-1}(1) LQ^{\ell-1} C,\end{aligned}$$

si la matrice Q vérifie $Q^\ell - a_1 Q^{\ell-1} - a_2 Q^{\ell-2} - \cdots - a_\ell I_N = 0$.

On demande d'écrire une procédure Maple qui prend en entrée une représentation linéaire et un nom x et fournit en sortie la série rationnelle associée sous forme d'une fraction rationnelle en x. On utilisera comme polynôme annulateur le polynôme caractéristique fourni par Maple dans le *package* `linalg`. À titre d'exemple la représentation linéaire

$$L = \begin{pmatrix} 0 & 2 \end{pmatrix}, \qquad Q = \begin{pmatrix} 3 & 3 \\ 1 & 5 \end{pmatrix}, \qquad C = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

définit la série

$$F(x) = LC + LQCx + LQ^2Cx^2 + LQ^3Cx^3 + \cdots$$

dont les premiers termes suivent.

```
> L:=array(1..1,1..2,[[0,2]]):
> Q:=array(1..2,1..2,[[3,3],[1,5]]):
> C:=array(1..2,1..1,[[1],[0]]):
> evalm(L &* C),seq(evalm(L &* Q^n &* C),n=1..5);
```

$$[0], [2], [16], [104], [640], [3872]$$

La procédure `linreptoseries` a l'effet demandé.

```
> linreptoseries(L,Q,C,x);
```

$$2\frac{x}{1-8x+12x^2}$$

```
> series(",x);
```

$$2x + 16x^2 + 104x^3 + 640x^4 + 3872x^5 + O(x^6)$$

Inversement on peut chercher une représentation linéaire d'une série rationnelle donnée sous la forme d'une fraction rationnelle qui n'admet pas le pôle 0. Il suffit pour cela d'utiliser une matrice compagne liée au dénominateur, ce qui revient à exprimer matriciellement une récurrence linéaire. MAPLE fournit la matrice compagne par la procédure `linalg/companion`. Il ne reste plus qu'à déterminer le vecteur ligne L et le vecteur colonne C pour construire une procédure `seriestolinrep` qui prend en entrée une fraction rationnelle qui n'admet pas 0 comme pôle et qui renvoie une représentation linéaire de cette fraction.

7.5. Vibrations d'une membrane. Les vibrations d'une membrane sont régies, dans de bonnes hypothèses de régularité, par l'équation aux dérivées partielles [7]

$$\frac{1}{\nu^2}\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}.$$

Nous envisageons ici deux cas : ou bien la membrane est rectangulaire et la fonction u est définie sur le rectangle $0 \le x \le A$, $0 \le y \le B$; ou bien la membrane est circulaire et la fonction u est définie sur le disque $r = \sqrt{x^2+y^2} \le R$. Dans chaque cas nous supposons que la condition aux limites est de type Dirichlet : la fonction u est nulle sur le pourtour de la membrane.

Dans le premier cas la solution stationnaire d'indice $(k,\ell) \in \mathbb{N}^* \times \mathbb{N}^*$ est définie par

$$\omega_{k,\ell} = \pi\sqrt{\frac{k^2}{A^2} + \frac{\ell^2}{B^2}},$$
$$u_{k,\ell}(x,y,t) = (a_{k,\ell}\cos(\omega_{k,\ell}\nu t) + b_{k,\ell}\sin(\omega_{k,\ell}\nu t))\sin\left(\frac{k\pi x}{A}\right)\sin\left(\frac{\ell\pi y}{B}\right)$$

avec $a_{k,\ell}$ et $b_{k,\ell}$ arbitraires. En utilisant la procédure `animate3d` du *package* `plots`, illustrer ce résultat.

Dans le second cas les solutions stationnaires qui ont la symétrie de révolution sont indexées par un entier naturel non nul n et s'écrivent

$$u_n(r,t) = \left(a_n\cos\left(\frac{\rho_n}{R}\nu t\right) + b_n\sin\left(\frac{\rho_n}{R}\nu t\right)\right)J_0\left(\frac{\rho_n r}{R}\right)$$

en notant J_0 la fonction de Bessel d'indice 0 et ρ_n son n-ième zéro. Les fonctions de Bessel de première espèce J_ν sont ontenues en MAPLE par la procédure `BesselJ`. On peut calculer numériquement ρ_n avec la procédure `fsolve`, sachant qu'il vaut à peu près $n\pi - \pi/4$ [1, formule 9.5.12]. D'autre part la procédure `display` du *package* `plots` permet de créer une animation en utilisant l'option `insequence=true`. Il suffit donc de créer une séquence d'images tridimensionnelles avec la commande `plot3d` pour voir vibrer une membrane circulaire. Précisons qu'il est plus adapté de définir les surfaces utilisées sous forme paramétrique avec des coordonnées cylindriques.

7.6. Des solutions. Nous apportons ici des solutions possibles aux problèmes proposés. Il est clair que le lecteur peut fixer un niveau de rigueur plus ou moins élevé ; il est cependant indispensable d'énoncer clairement et en premier lieu la spécification du programme cherché, c'est-à-dire les paramètres pris en entrée et le résultat fourni en sortie. Nous nous sommes astreints à typer tous les paramètres des procédures vues par un éventuel utilisateur ; on peut évidemment commencer par écrire des versions plus molles sans vérification de types.

Algorithme boustrophédon. L'algorithme boustrophédon peut être mis en pratique de la façon suivante. On voit les valeurs comme les éléments d'un tableau, de la même manière que dans l'énoncé. Les lignes sont numérotées à partir de 0 et dans la ligne n l'indice r va de 0 à n, tout comme pour le triangle de Pascal. Le procédé de construction se traduit tout de suite par une procédure récursive, que l'on rend plus efficace par l'usage de l'option remember.

```
boustrophedon:=proc(n,r)
# L'instruction boustrophedon(n,r) renvoie l'element de la ligne n, colonne r du tableau
 option remember;
  if (n=0 and r=0) then 1 # on initialise
  elif type(n,even) then # si n est pair on demarre a l'extremite droite
    if r=n then 0 else boustrophedon(n,r+1)+boustrophedon(n-1,r) fi
  else  # si n est impair on demarre a l'extremite gauche
    if r=0 then 0 else boustrophedon(n,r-1)+boustrophedon(n-1,r-1); fi
  fi
end:
```

Cette procédure permet de calculer les éléments du tableau.

```
> for n from 0 to 10 do seq(boustrophedon(n,r),r=0..n) od;
```

$$1$$
$$0, 1$$
$$1, 1, 0$$
$$0, 1, 2, 2$$
$$5, 5, 4, 2, 0$$
$$0, 5, 10, 14, 16, 16$$
$$61, 61, 56, 46, 32, 16, 0$$
$$0, 61, 122, 178, 224, 256, 272, 272$$
$$1385, 1385, 1324, 1202, 1024, 800, 544, 272, 0$$
$$0, 1385, 2770, 4094, 5296, 6320, 7120, 7664, 7936, 7936$$
$$50521, 50521, 49136, 46366, 42272, 36976, 30656, 23536, 15872, 7936, 0$$

Cependant on remarque qu'il n'y aucune vérification sur les arguments de la procédure. On type donc les arguments et au sein de la procédure on vérifie l'inégalité $n \geq r$.

```
boustrophedon:=proc(n:nonnegint,r:nonnegint)
  if args[1]>=args[2] then # dans une procedure args est la sequence
    boustrophedonbis(n,r)  # des arguments
  else ERROR(`boustrophedon expects its 1st argument, n, to be not greater
               than its second argument, r, but received`,n,r)
  fi
end:
```

Il est stupide de vérifier sans cesse le type des arguments dans les appels récursifs. On utilise donc une procédure auxiliaire `boustrophedonbis`, qui fonctionne comme la première que nous avons écrite, sans vérification de type. De plus c'est elle qui doit être munie de l'option remember.

```
boustrophedonbis:=proc(n,r)
  option remember;
```

```
  if (n=0 and r=0) then 1
  elif type(n,even) then
    if r=n then 0 else boustrophedonbis(n,r+1)+boustrophedonbis(n-1,r) fi
  else
    if r=0 then 0 else boustrophedonbis(n,r-1)+boustrophedonbis(n-1,r-1)fi
  fi
end:
```

Il ne reste plus qu'à écrire les deux procédures qui fournissent les développements de tangente et sécante.

```
tanseries:=proc(x:name,n:nonnegint)
  local i;
  series(convert([seq(boustrophedon(i,i)/i!*x^i,i=1..n-1)],`+`),x,n)
end:
secseries:=proc(x:name,n:nonnegint)
  local i;
  series(convert([seq(boustrophedon(i,0)/i!*x^i,i=0..n-1)],`+`),x,n)
end:
```

L'emploi de `series` est là dans un souci de compatibilité; son faible coût sur un polynôme excuse cette coquetterie.

```
> tanseries(x,18);
```

$$x+\frac{1}{3}x^3+\frac{2}{15}x^5+\frac{17}{315}x^7+\frac{62}{2835}x^9+\frac{1382}{155925}x^{11}+\frac{21844}{6081075}x^{13}+\frac{929569}{638512875}x^{15}+\frac{6404582}{10854718875}x^{17}$$

Développement d'une fonction implicite. Passons à la recherche du développement en série d'une fonction implicite, dans les conditions du théorème des fonctions implicites. L'expression de fonction passée en paramètre est typée par le type `algebraic`. On applique la méthode indiquée dans le texte après avoir vérifié les conditions d'application du théorème.

```
newtondev:=proc(F:algebraic,x:name,y:name,n:nonnegint)
# F est une expression qui definit une fonction de deux
# variables x et y, n l'ordre du developpement en serie.

  local F1,Z,p;
  F1:=diff(F,y);
  # il faut d'abord verifier les hypotheses du theoreme des
  # fonctions implicites
  if ( eval(subs(x=0,y=0,F))<>0 or eval(subs(x=0,y=0,F1))=0 )
    then ERROR(`Singularite en 0`) fi;
  Z:=0; # on initialise l'iteration
  for p while 2^p<n do
    Z:=eval(Z-subs(y=Z,F)/subs(y=Z,F1));
    # on tronque le developpement
    # on convertit en polynome pour ne pas propager le O(x^(2^p))
    Z:=expand(convert(series(Z,x,2^p),polynom));
    # la commande expand aide series a donner un resultat
    # correct meme dans le cas ou F est de type series,
    # ce qui est autorise par le type algebraic.
  od;
  # il faut iterer encore une fois le processus, en se limitant
  # cette fois a l'ordre n.
  Z:=eval(Z-subs(y=Z,F)/subs(y=Z,F1));
  series(expand(convert(series(Z,x,n),polynom)),x,n);
end:
```

Testons ceci sur un exemple

```
> G:=exp(-a*x)-1+y*cos(x)+y^2*sin(x);
```

$$e^{-ax} - 1 + y\cos(x) + y^2\sin(x)$$

```
> newtondev(G,x,y,6):
> map(normal,");
```

$$ax - \frac{1}{2}a^2x^2 + \left(\frac{1}{6}a^3 + \frac{1}{2}a - a^2\right)x^3 + \left(a^3 - \frac{1}{24}a^4 - \frac{1}{4}a^2\right)x^4 + \left(-\frac{4}{3}a^2 - \frac{7}{12}a^4 + \frac{25}{12}a^3 + \frac{5}{24}a + \frac{1}{120}a^5\pi\right)x^5 + O\left(x^6\right)$$

Donnons un exemple classique un peu plus subtil. L'équation $\tan(cx) = 1/x$ admet pour tout entier k assez grand une racine z_k proche de $k\pi/c$. Pour se ramener à l'origine, on pose $t = 1/k$ et $z_k = k\pi/c + x$. Les deux variables t et x sont reliées par l'équation $L(t, x) = 0$ qui suit.

```
> L:=c*x-arctan(1/(1/t*Pi/c+x));
```

$$L := cx - \arctan\left(\frac{1}{\frac{\pi}{tc} + x}\right)$$

On applique d'abord `normal` pour que la substitution $(t, x) = (0, 0)$ ne produise pas d'erreur.

```
> newtondev(map(normal,L),t,x,6);
```

$$\pi^{-1}t + \left(-\frac{c}{\pi^3} - \frac{c^2}{3\,\pi^3}\right)t^3 + \left(\frac{c^4}{5\,\pi^5} + \frac{2\,c^2}{\pi^5} + \frac{4\,c^3}{3\,\pi^5}\right)t^5 + O\left(t^7\right)$$

Suite de Sturm. La procédure `dg_sturm` s'applique à des polynômes à coefficients rationnels, libres de carré et non constants. La comparaison de deux rationnels est décidable ; en particulier le signe d'un rationnel est décidable, contrairement au signe d'un flottant ; c'est pourquoi on utilise sans cesse les rationnels dans cet exercice. La procédure `dg_sturm` traduit exactement le calcul de la suite de Sturm et des suites de changement de signes décrit dans le texte.

```
dg_sturm:=proc(P,x:name,interv:range(rational))
# renvoie le nombre de racines de P(x) dans l'intervalle ]a,b]
# si interv=a..b
local a,b,p,q,r,Va,Vb,i,j;
  if not type(P,polynom(rational,x)) or has(gcd(P,diff(P,x)),x) then
      ERROR(`invalid arguments`) fi; # on verifie le typage
  a:=op(1,interv); # on nomme les extremites
  b:=op(2,interv); # de l'intervalle
  p:=expand(P); # pour la suite, il nous faut la forme developpee de P
  q:=diff(p,x);
  Va[0]:=signum(subs(x=a,p));
  Vb[0]:=signum(subs(x=b,p));
  for i while degree(q,x)>0 do
    Va[i]:=signum(subs(x=a,q)); # on stocke les signes dans
    Vb[i]:=signum(subs(x=b,q)); # des tables
    r:=-rem(p,q,x);
    p:=q; q:=r;
  od;
  Va[i]:=signum(q);
  Vb[i]:=signum(q);
  var_signe([seq(Va[j],j=0..i)])-var_signe([seq(Vb[j],j=0..i)]);
end:
```

La procédure **var_signe** travaille sur des arguments non typés car elle n'est pas vue par l'utilisateur.

```
var_signe:= proc(V) # renvoie le nombre de changements de signe
  local W,i;        # dans la liste de signe V
  W:=subs(0=NULL,V); # on commence par enlever les zeros
  W:=[seq(op(i,W)*op(i+1,W),i=1..nops(W)-1)];
  # "changement de signe" equivaut a "produit negatif"
  nops(select(type,W,negative))
end:
```

La procédure **dg_sturm** étant disponible, on peut déterminer des intervalles d'extrémités rationnelles qui contiennent les racines réelles d'un polynôme à coefficients rationnels libre de carré. Pour cela on part d'un intervalle à extrémités entières qui contient toutes les racines réelles, puis on procède par dichotomie en évacuant les intervalles qui ne contiennent pas de racine, ce qui est possible puisque **dg_sturm** permet de compter le nombre de racines dans un intervalle donné. On itère la dichotomie jusqu'à ce que les intervalles aient une longueur inférieure à une précision donnée. On renvoie alors un ensemble de couples $[I, \nu]$ où I est un intervalle $]a, b]$ à extrémités rationnelles et ν est le nombre de racines du polynôme dans l'intervalle.

```
readlib(iroot):

real_zeroes:= proc(P,x:name,eps:positive)
local Q,R,n,cd,i;
  if not type(P,polynom(rational,x)) or has(gcd(P,diff(P,x)),x) then
    ERROR(`invalid arguments`) fi;
  Q:=expand(P);
  n:=degree(Q,x);
  cd:=abs(coeff(Q,x,n));
  # on utilise la majoration des valeurs absolues des racines de Q
  R:=2*max(seq( 1+iroot(ceil(abs(coeff(Q,x,n-i))/cd),i),i=1..n));
  dichotomie(Q,x,eps,-R..R)
end:
```

On s'astreint à travailler avec des entiers ou des rationnels pour garantir les comparaisons. Pour cela on utilise la procédure **iroot**, qui fournit les racines n-ièmes entières et approximatives. La valeur affectée à la variable R dans la procédure précédente peut certainement être améliorée.

```
dichotomie:= proc(P,x,eps,interv)
  # renvoie un ensemble d'intervalles [a,b[ de longueur < eps
  # et contenant les racines reelles de P
  local a,b,n;
  n:=dg_sturm(P,x,interv);
  if n=0 then RETURN({}) fi; # pas de racines
  a:=op(1,interv); b:=op(2,interv);
  if b-a<eps then RETURN({[interv,n]}) fi;
  # on s'arrete lorsque la taille de l'intervalle est < eps
  # sinon on divise en deux et on recolle les morceaux
  dichotomie(P,x,eps,a..(a+b)/2) union dichotomie(P,x,eps,(a+b)/2..b)
end:
```

Il ne reste plus qu'à tester cette procédure sur quelques exemples comme le suivant.

```
> P:=convert(series(sin(x),x,10),polynom);
```

$$P := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9$$

```
> real_zeroes(P,1/10^6);
```

$$\left\{[\frac{83267883}{16777216}..\frac{20816973}{4194304},1],[-\frac{20816973}{4194304}..-\frac{83267883}{16777216},1],[\frac{52826247}{16777216}..\frac{3301641}{1048576},1]\right.,$$
$$\left.[-\frac{9}{16777216}..0,1],[-\frac{3301641}{1048576}..-\frac{52826247}{16777216},1]\right\}$$

```
> evalf(");
```

$$\{[4.963152587..4.963153124,1.0],[-4.963153124..-4.963152587,1.0],$$
$$[3.148689687..3.148690224,1.0],[-0.0000005364418030..0,1.0],$$
$$[-3.148690224..-3.148689687,1.0]\}$$

La procédure `realroot` a un usage similaire à notre `real_zeroes`.

Séries rationnelles. La procédure `linreptoseries` qui suit utilise le *package* d'algèbre ; on suppose donc que l'on a défini les noms courts de ce *package* par `with(linalg)`. D'autre part les trois arguments L, Q, C sont liés ; on vérifie donc la compatibilité des données par un type procédural qui va s'appliquer à la liste $[L,Q,C,x]$.

```
`type/linreptoseriesarg`:=proc(arg)
    evalb(nops(arg)=4
      and rowdim(arg[1])=1
      and coldim(arg[1])=rowdim(arg[2])
      and rowdim(arg[2])=coldim(arg[2])
      and coldim(arg[2])=rowdim(arg[3])
      and coldim(arg[3])=1
      and type(arg[4],name))
end:
```

On se contente ensuite d'appliquer les formules données dans le texte.

```
linreptoseries:=proc(L,Q,C,x)
 local n,d,i,c,chi,deg,q;
 if type([args],linreptoseriesarg) then
   # on calcule le polynome caracteristique de Q et
   # on en tire un denominateur pour la fraction
   # en passant au polynome reciproque
   chi:=charpoly(Q,x);
   deg:=degree(chi,x);
   d:=collect(expand(x^deg*subs(x=1/x,chi)),x);
   # on calcule le numerateur
   c[0]:=coeff(d,x,0);
   q[0]:=C;
   for i to deg-1 do
     c[i]:=c[i-1]+coeff(d,x,i)*x^i;
     q[i]:=multiply(Q,q[i-1])
   od;
   n:=convert([seq(x^(deg-1-i)*c[i]*multiply(L,q[deg-1-i])[1,1],i=0..deg-1)],`+`);
   # on renvoie la fraction
   normal(n/d)
 fi
end:
```

Dans l'autre sens on applique l'idée suivante : l'opérateur de décalage se traduit sur les séries en l'opérateur

$$F(x)\longmapsto\frac{F(x)-F(0)}{x}.$$

On construit donc la suite (H_k) définie par

$$H_0(x) = F(x), \qquad H_{k+1}(x) = \frac{H_k(x) - H_k(0)}{x}$$

et on cherche la première relation de dépendance linéaire

$$c_0 H_0 + \cdots + c_d H_d = 0\,;$$

la représentation linéaire est alors

$$L = (\ H_0(0) \quad \ldots \quad H_{d-1}(0)\ ), \qquad Q := \begin{pmatrix} 0 & 0 & 0 & \ldots & -c_0/c_d \\ 1 & 0 & 0 & & -c_1/c_d \\ 0 & 1 & 0 & & \\ & & & & \\ 0 & & & 1 & -c_{d-1}/c_d \end{pmatrix}, \qquad C = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

D'autre part on vérifie la correction des données en deux temps ; on utilise d'abord le type **ratpoly** qui exprime ce qu'est une fraction rationnelle, puis dans la procédure on vérifie que 0 n'est pas pôle de la fraction.

```
seriestolinrep:=proc(F:ratpoly,x:name)
  local f,H,i,S,j,sys,inc,sol,k,L,Q,C;
  f:=normal(F);
  if f=0 then L:=array(1..1,1..1,[[0]]);
              Q:=array(1..1,1..1,[[0]]);
              C:=array(1..1,1..1,[[1]]);
  elif subs(x=0,denom(f))<>0 then
    H[0]:=f;
    for i to max(degree(numer(f),x),degree(denom(f),x))+1 do
      H[i]:=normal((H[i-1]-subs(x=0,H[i-1]))/x);
      S:=expand(numer(normal(convert([seq(c[j]*H[j],j=0..i)],`+`))));
      sys:={seq(coeff(S,x,j),j=0..degree(S,x))};
      inc:={seq(c[j],j=0..i)};
      sol:=solve(sys,inc);
      # des qu'il y a dependance lineaire on sort de la boucle
      if (sol <> {seq(c[j]=0,j=0..i)}) then break fi;
    od;
    C:=array(1..i,1..1,sparse);
    C[1,1]:=1;
    L:=array(1..1,1..i,[[seq(subs(x=0,H[k]),k=0..i-1)]]);
    Q:=array(1..i,1..i,sparse);
    for k from 2 to i do Q[k,k-1]:=1 od;
    for k from 1 to i do Q[k,i]:=-c[k-1]/c[i] od;
    Q:=subs(sol,eval(Q));
  op(L),op(Q),op(C);
  else
    ERROR(`denominator must not be 0 at 0`)
  fi;
end:
```

Il est clair que la recherche d'une relation de dépendance entre les H_k dans la procédure précédente ne fournit rien d'autre qu'une certaine matrice compagne. Nous laissons au lecteur le soin de simplifier la procédure en tenant compte de cette remarque.

```
> seriestolinrep((1-x)/(1+x-x^5),x);
```

$$\left[\begin{array}{ccccc} 1 & -2 & 2 & -2 & 2 \end{array}\right], \quad \left[\begin{array}{ccccc} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{array}\right], \quad \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right]$$

```
> linreptoseries(",x);
```

$$\frac{x-1}{-1-x+x^5}$$

Membranes vibrantes. Ce dernier exercice illustre les possibilités graphiques de MAPLE mais ne ressortit pas au calcul formel. Dans chaque cas le typage est évident et on applique simplement les formules données dans le texte. Dans le cas du rectangle on utilise la procédure `animate3d` ; le paramètre `frames` détermine le nombre d'images de l'animation. Pour le disque on a préféré la procédure `plots/display` avec l'option `insequence=true` car l'option `coords=cylindrical` de la procédure `plot3d` attend une surface dont l'équation est sous la forme $r = F(\vartheta, z)$ et non $z = F(r, \vartheta)$.

```
rect:=proc(A:positive,B:positive,nu:positive,k:posint,l:posint,a:numeric,b:numeric)
 local omega,z;
  omega:=Pi*sqrt(k^2/A^2+l^2/B^2);
  z:=omega*nu*t;
  plots[animate3d]((a*cos(z)+b*sin(z))*sin(k*Pi*x/A)*sin(l*Pi*y/B),
                   x=0..A,y=0..B,t=0..2*Pi/omega/nu,frames=20);
end:
```

On remarquera dans la procédure `disk` le calcul numérique du n-ième zéro de la fonction de Bessel J_0 ; il est obtenu en précisant dans `fsolve` un intervalle où chercher ce zéro.

```
disk:=proc(R:positive,nu:positive,n:posint,a:numeric,b:numeric)
  local rho,omega,z,k,nbre;
  nbre:=20;
  rho:=fsolve(BesselJ(0,x),x,-Pi/4+n*Pi-1..-Pi/4+n*Pi+1);
  omega:=rho/R;
  z:=omega*nu*2*k*Pi/omega/nu/nbre;
  plots[display]([seq(
     plot3d([r*cos(theta),r*sin(theta),(a*cos(z)+b*sin(z))*BesselJ(0,omega*r)],
             r=0..R,theta=0..2*Pi),k=0..nbre)],insequence=true);
end:
```

La mise en pratique des ces deux procédures dépend beaucoup de la machine utilisée ; pour une machine peu puissante il faudrait limiter le nombre de dessins, qui est ici fixé à vingt.

8. Conclusion

Il est clair que ce petit texte ne saurait donner une présentation exhaustive de MAPLE. Beaucoup de points y sont passés sous silence et les sujets traités ne sont pas approfondis. Rappelons que MAPLE comporte actuellement plus de 2500 procédures. Pour aller plus loin, il faut évidemment pratiquer. Dans la mesure où toute l'information nécessaire est contenue dans le *help* nous ne pouvons qu'insister sur le point suivant.

Il ne faut pas hésiter à lire les pages d'aide.

On remarquera que chaque page d'aide comprend une description, une suite d'exemples et une liste de sujets connexes. Cette dernière ligne de la page ne doit pas être négligée car elle permet de découvrir de nouvelles procédures dont on n'avait pas soupçonné l'existence ou simplement d'atteindre le sujet cherché.

On peut se cultiver en compulsant quelques ouvrages sur le calcul formel. Le livre de J. Davenport, Y. Siret et E. Tournier, *Calcul formel*, Masson, 1993, collection *Études et recherche en informatique*, est particulièrement agréable à lire et présente les problèmes essentiels du calcul formel. Pour approfondir le sujet, un livre plus technique qui permet de comprendre les algorithmes employés est *Algorithms for Computer Algebra* de K. O. Geddes, S. R. Czapor et G. Labahn, Kluwer Academic Publisher, 1992.

Pour ce qui est du langage MAPLE lui-même, les ouvrages fournis avec le logiciel (introduction, manuels de référence du langage et des bibliothèques, mises à jour des versions V.2 et V.3) sont utiles ; une bonne partie de ces textes se retrouve dans le *help*. Le livre de A. Levine, *Introduction à Maple*, Édition Marketing (ellipses), 1994, n'apporte pas d'informations supplémentaires par rapport aux feuilles d'aide mais peut être agréable pour un débutant peu familier avec la langue anglaise ; cependant certaines de ses assertions sont critiquables. L'ouvrage *Premiers pas en Maple* de P. Fortin et R. Pomès, Vuibert 1994, est susceptible du même commentaire d'autant plus qu'il développe par endroits une vision qui ne correspond pas à l'esprit du logiciel.

Certains auteurs font découvrir des mathématiques au travers de MAPLE. On peut citer *Maple via Calculus, A Tutorial Approach* par R.J. Lopez, Birkhäuser, 1994 ; *Linear Algebra with Maple V*, E.W. Johnson, Brooks/Cole, 1993; ou encore *Linear Algebra with Maple*, W.C. Bauldry, B. Evans, J.Johnson, John Wiley & Sons, 1995.Tous ces ouvrages restent superficiels et n'ont pas pour but d'initier au fonctionnement de MAPLE. Quant à *First Steps in Maple*, W. Burkhard, Springer-Verlag, 1994, il est excessivement élémentaire à tous points de vue.

Trois ouvrages sur MAPLE paraissent excellents. Le livre de A. Heck, *Introduction to Maple*, Springer-Verlag, 1993, donne quelques notions de calcul formel, présente largement MAPLE et donne des applications relativement originales et tournées vers la physique. Celui de R.M. Corless, *Essential Maple*, Springer-Verlag, 1995,malgré son bas prix donne une vision concise, claire et profonde de MAPLE appuyée d'exemples. Enfin, celui de C. Gomez, B. Salvy et P. Zimmermann, *Calcul formel, Mode d'emploi. Exemples en Maple*, Collection Logique Mathématiques Informatique, Masson, 1995, présente MAPLE de façon approfondie. Surtout il fournit de nombreux exemples mathématiques très intéressants et fait bien voir comment utiliser au mieux MAPLE. Signalons aussi le compendium de D. Redfern, *The Maple Handbook*, Springer-Verlag, 1994, qui commente tous les noms réservés de MAPLE.

Terminons par l'ouvrage extrêmement intéressant de W. Gander et J. Hřebíček, *Solving Problems in Scientific Computing Using Maple and Matlab*, Springer-Verlag, 1993, qui propose dix-neuf thèmes de mathématiques ou de physique traités en détail, mais n'est pas un ouvrage d'initiation à MAPLE.

RÉFÉRENCES

1. Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions.* Dover, 1973. A reprint of the tenth National Bureau of Standards edition, 1964.
2. M. D. Atkinson. How to compute the series expansions of $\sec x$ and $\tan x$. *American Mathematical Monthly*, pages 387–389, May 1986.
3. Peter Henrici. *Applied and Computational Complex Analysis.* John Wiley, New York, 1974.
4. D. E. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms. Addison-Wesley, 2nd edition, 1981.
5. D. P. Parent, D. Barsky, F. Bertrandias, G. Christol, A. Decomps, H. Delange, J.-M. Deshouillers, K. Gérardin, J. Lagrange, J.-L. Nicolas, M. Pathiaux, G. Rauzy, and M. Waldschmidt. *Exercices de théorie des nombres.* μ_B. Gauthiers-Villars, 1978.
6. R. Kalaba R. Bellman. *Selected Papers on Mathematical Trends in Control Theory*, chapter 4. Dover Publications, 1964. On the conditions under which an equation has only roots with negative real parts, Translation from A. Hurwitz, Über die Bedingungen unter welchen eine Gleichung nur Wurzeln mit negativen reellen Theilen besitzt, Mathematische Annalen, Vol. 46, 1895, pp. 273–284.
7. Laurent Schwartz. *Méthodes mathématiques pour les sciences physiques.* Hermann, 1965.
8. G. Viennot. Quelques algorithmes de permutation. In *Astérisque*, pages 38–39, 275–293. Société Mathématique de France, 1976.